Application of Convolutional Neural Networks in Inverse Problems of Ocean Acoustics

Vassilios Tzirakis

UNIVERSITY OF CRETE

Department of Mathematics and Applied Mathematics

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ
Τμήμα Μαθηματικών και Εφαρμοσμένων Μαθηματικών

Application of Convolutional Neural Networks in Inverse Problems of Ocean Acoustics

Vassilios Tzirakis

Διπλωματική εργασία υποβληθείσα προς μερική εκπλήρωση
των απαραίτητων προϋποθέσεων για την απόκτηση του
Μεταπτυχιακού Διπλώματος Ειδίκευσης
στα Εφαρμοσμένα Μαθηματικά

16 Νοεμβρίου 2023

Επιβλέπων:

Μιχαήλ Ταρουδάκης

Μέλη επιτροπής:

Μιχαήλ Ταρουδάκης

Παναγιώτης Χατζηπαντελίδης

Γεώργιος Μακράκης

Ημερομηνία εξέτασης: 16 Νοεμβρίου 2023

Preface

The thesis main inspiration is the research on physical problems that occur in our daily lives. In particular, I am really interested in wave propagation in ocean acoustics, as its theory is applied to many environmental problems, a vast majority of which are inverse problems. My interest in this thesis, is the examination of a particular inverse problem, with the help of Convolutional Neural Networks techniques that are increasingly used in every day applications. There are no prerequisites for one to understand the work that its done, as I try to explain almost every concept from scratch. Nevertheless, I suggest that the reader is a little bit familiar with the concept of ocean acoustics modeling theory and the main functionality of the structure of Neural Networks, in order to have a more pleasant reading of the thesis.

Acknowledgements

I would like to thank both Prof. Michael Taroudakis and Dr. Costas Smaragdakis for providing their important knowledge and instructions to help me complete this thesis. I also would like to thank everyone who has supported me in the challenging years of academic studying and exceptionally my beloved grandfather Grigoris.

Contents

List of Figures

List of Tables

Περίληψη

Κύριος στόχος της διπλωματικής εργασίας είναι η μελέτη της εφαρμογής ενός Συνελικτικού Νευρωνικού Δικτύου για την εκτίμηση των καμπυλών διασποράς που χαρακτηρίζουν το φασματογράφημα ενός ακουστικού σήματος που καταγράφεται σε έναν θαλάσσιο κυματοδηγό. Οι καμπύλες διασποράς αποτελούν δεδομένα εισόδου για μια κατηγορία αντίστροφων προβλημάτων στην υποβρύχια ακουστική, τα οποία αντιμετωπίζονται χρησιμοποιώντας την ανάλυση στο χώρο χρόνου-συχνότητας ενός καταγεγραμμένου σήματος. Για να κάνω μια συνοπτική παρουσίαση της δουλειάς μου, η διατριβή ξεκινά με μια σύντομη παρουσίαση της κυματικής θεωρίας με ειδική αναφορά στη μοντελοποίηση μιας ευρυζώνιας πηγής. Το Κεφάλαιο 2 είναι αφιερωμένο στον μετασχηματισμό Fourier καθώς είναι και το κύριο εργαλείο για την επεξεργασία σήματος. Το φασματόγραμμα παρουσιάζεται σε αυτό το κεφάλαιο. Καθώς οι εφαρμογές που εξετάζονται στη διατριβή σχετίζονται με αντίστροφα προβλήματα, μια εισαγωγή στη θεωρία των αντίστροφων προβλημάτων παρουσιάζεται στο Κεφάλαιο 3. Στο Κεφάλαιο 4, παρουσιάζονται τα βασικά χαρακτηριστικά ενός Συνελικτικού Νευρωνικού Δικτύου (CNN) που είναι το κύριο εργαλείο που χρησιμοποιείται για την εξαγωγή πληροφοριών για τις καμπύλες διασποράς. Τέλος, το Κεφάλαιο 5 παρουσιάζει μια δοκιμή, σχετικά με την εφαρμογή του CNN, σε ένα προσομοιωμένο υποβρύχιο ακουστικό σήμα, το οποίο έχει τη μορφή σημάτων που χρησιμοποιούνται σε εφαρμογές ακουστικής τομογραφίας ή αναγνώρισης πυθμένα. Τα συμπεράσματα της μελέτης μου παρουσιάζονται στο τελευταίο κεφάλαιο.

Abstract

The main goal of the thesis is to study the application of a Convolutional Neural Network for the estimation of the dispersion curves that characterize the spectrogram of an acoustic signal recorded in an ocean waveguide. Dispersion curves are input data for a class of inverse problems in underwater acoustics which are treated using the time frequency analysis of a recorded signal. To make a concise presentation of my work, the thesis starts with a brief presentation of wave theory with a special reference to modeling a broadband source. Chapter 2 is devoted to Fourier transform as it is the main tool for signal processing in my work. The spectrogram is introduced in this chapter. As the applications considered in the thesis are related to inverse problems, an introduction to the theory of inverse problems is is presented in Chapter 3. In Chapter 4, the basic features of a Convolutional Neural Network (CNN), which is the main tool used to extract the information on the dispersion curves, are presented. Finally Chapter 5 presents a test case on the application of the CNN in a simulated underwater acoustic signal, which is of the form of signals used in applications of ocean acoustic tomography or seabed classification. The conclusions of my study are presented in the final chapter.

CHAPTER 1

Wave Theory on Ocean Acoustics

1.1 Forward Problem

The goal of a *forward problem* in ocean acoustics is to solve the *acoustic wave equation* in a waveguide. A typical vertical slice of the environment is shown in figure 1.1:

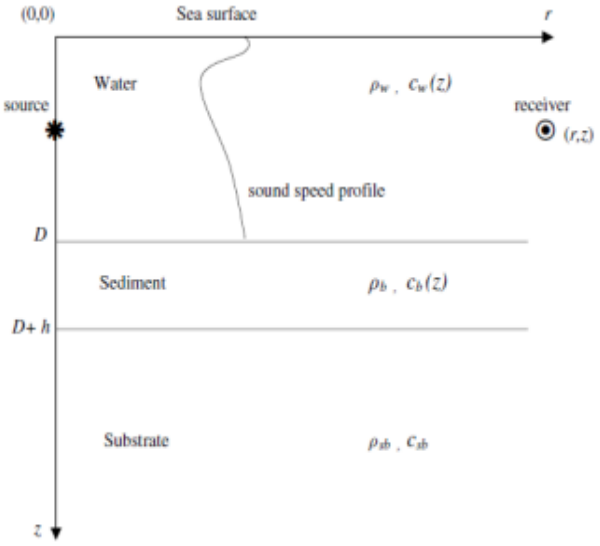


Figure 1.1: A vertical slice of a typical oceanic waveguide $[0,\infty)\times[0,\infty)$ (obtained from the thesis of Savinas Antoulinaki : Statistical characterization of signals using wavelet packets)

In the environment shown in figure 1.1 the layers defining the domain of propagation are three.The first one is water of thickness *D*, with density ρ_w and characterized by an acoustic velocity profile, that depends only on depth *z*, *c(z)*.The second is a fluid sediment of thickness *h* with density ρ_b and acoustic velocity profile $c_b(z)$ and the third one is a semi-infinite fluid substrate with density ρ_{sb} and acoustic velocity profile c_{sb}. We consider a *monochromatic* source at depth z_0 and the receiver of the signal at depth z_r and range *r* from the source. Note that this is an example of an oceanic waveguide, of the form used in this thesis. More general environments with complicated boundary interfaces and/or involving elastic domains are outside the scope of this work.

1.2 The acoustic wave equation and boundary conditions

1.2.1 General Coordinates

The goal of an acoustic propagation problem is to calculate the acoustic field (*pressure P*) as a function of position and time when an acoustic source of known characteristics emits an acoustic signal in the environment. The linearized acoustic wave equation for the acoustic pressure is given by:

$$\nabla^2 P(\mathbf{x},t) - \frac{1}{\rho(\mathbf{x})}\nabla\rho(\mathbf{x})\cdot\nabla P(\mathbf{x},t) - \frac{1}{c^2}\frac{\partial^2 P(\mathbf{x},t)}{\partial t^2} = -S(\mathbf{x},\mathbf{x}_0,t) \tag{1.1}$$

where c is the sound velocity at the corresponding domain which in general and for range-independent environments which are the class of environments studied here, $c = c(\mathbf{x})$ and $\rho(\mathbf{x})$ is the density of the acoustic medium.

The acoustic source to be considered here is point harmonic of the form:

$$S(\mathbf{x},\mathbf{x}_0,t) = A\delta(\mathbf{x}-\mathbf{x}_0)e^{-i\omega t} \tag{1.2}$$

Note that $\rho(\mathbf{x})$ comes from *Euler's Equation* from fluid mechanics[2]

Applying variable separation $P(\mathbf{x},t) = p(\mathbf{x})e^{-i\omega t}$, (1.1) we obtain the non-homogeneous *Helmholtz* equation:

$$\nabla^2 p(\mathbf{x}) - \frac{1}{\rho(\mathbf{x})}\nabla\rho(\mathbf{x})\cdot\nabla p(\mathbf{x}) + k^2(\mathbf{x})p(\mathbf{x}) = -\delta(\mathbf{x}-\mathbf{x}_0)^1 \tag{1.3}$$

1.2.2 Cylindrical Coordinates

The Helmholtz equation in underwater acoustics is typically solved in cylindrical coordinates. Let's transform the *Helmholtz* equation (1.3) to cylindrical coordinates. Assuming $\rho(\mathbf{x}) = \rho(z)$ and $c(\mathbf{x}) = c(z)$ which is a realistic assumption in horizontally stratified media we get:

$$\frac{1}{r}\frac{\partial}{\partial r}(r\frac{\partial p(r,z)}{\partial r}) + \frac{1}{r^2}\frac{\partial^2 p(r,z)}{\partial \phi^2} + \frac{\partial^2 p(r,z)}{\partial z^2} - \frac{1}{\rho(z)}\frac{d\rho(z)}{dz}\frac{\partial p(r,z)}{\partial z} + k(z)^2 p(r,z) = -\frac{1}{r}\delta(r-r_0)\delta(\phi-\phi_0)\delta(z-z_0) \tag{1.4}$$

Note that this assumption is consistent with the description of the waveguide presented in the previous paragraph[2]. In addition, we are going to deal with the case that p does not depend on angle ϕ, that is we have *axially symmetric environment*. Assuming that the source is at $r_0 = 0$ one can show that (1.4) can be written as:

$$\frac{\partial^2 p(r,z)}{\partial r^2} + \frac{1}{r}\frac{\partial p(r,z)}{\partial r} + \frac{\partial^2 p(r,z)}{\partial z^2} - \frac{1}{\rho(z)}\frac{d\rho(z)}{dz}\frac{\partial p(r,z)}{\partial z} + k(z)^2 p(r,z) = -\frac{1}{2\pi r}\delta(r)\delta(z-z_0) \tag{1.5}$$

1.2.3 Boundary conditions

In order to determine a *unique* solution of the acoustic problem, we should set boundary conditions based on the physics that we encounter on our *range-independent* environment. First, we assume that we do not have perturbation of the sound on the surface, that is the pressure at height zero is equal to zero. Furthermore, in order to preserve the physical convex structure between each layer and that they do not separate, we have to ensure that our function $p(r,z)$ is continuous on the interfaces as well as the vertical component of the velocity of sound particles. We also consider that no energy returns from infinity, $r \to \infty$ (*Sommerfeld condition*) and that for large depths, $z \to \infty$, the pressure tends to zero. Finally,

[1] For simplicity we choose $A = 1$

[2] Note that until (1.3) we have not used this assumption

we set constant density for each layer. The mathematical formulation of the previous, considering the symbolization:

$$\rho(z) = \begin{cases} \rho_1, & \text{if } 0 \leq z \leq D \\ \rho_2, & \text{if } D < z \leq D + h \\ \rho_3, & \text{if } z > D + h \end{cases}$$

$$p(r,z) = \begin{cases} p^{(1)}(r,z), & \text{if } 0 \leq z \leq D \\ p^{(2)}(r,z), & \text{if } D < z \leq D + h \\ p^{(3)}(r,z), & \text{if } z > D + h \end{cases}$$

is the following:

$$p^{(1)}(r,0) = 0 \tag{1.6}$$

$$\lim_{z\to D^-} p^{(1)}(r,z) = \lim_{z\to D^+} p^{(2)}(r,z) \tag{1.7}$$

$$\lim_{z\to D^-} \frac{1}{\rho_1}\frac{\partial p^{(1)}(r,z)}{\partial z} = \lim_{z\to D^+} \frac{1}{\rho_2}\frac{\partial p^{(2)}(r,z)}{\partial z} \tag{1.8}$$

$$\lim_{z\to (D+h)^-} p^{(2)}(r,z) = \lim_{z\to (D+h)^+} p^{(3)}(r,z) \tag{1.9}$$

$$\lim_{z\to (D+h)^-} \frac{1}{\rho_2}\frac{\partial p^{(2)}(r,z)}{\partial z} = \lim_{z\to (D+h)^+} \frac{1}{\rho_3}\frac{\partial p^{(3)}(r,z)}{\partial z} \tag{1.10}$$

$$\lim_{r\to\infty} \sqrt{r}\frac{\partial p(r,z)}{\partial r} - ik(z)p(r,z) = 0 \tag{1.11}$$

$$\lim_{z\to\infty} p^{(3)}(r,z) = 0 \tag{1.12}$$

1.3 Normal Mode Theory

1.3.1 Representation Theorem

Let us consider the homogeneous form of (1.5).By performing separation of variables $p(r,z) = \Phi(r)u(z)$ we obtain the system of ordinal differential equations:

$$\frac{d^2\Phi(r)}{dr} + \frac{1}{r}\frac{d\Phi(r)}{dr} + \lambda\Phi(r) = 0 \tag{1.13}$$

$$\frac{d}{dz}\left(\frac{1}{\rho(z)}\frac{du(z)}{dz}\right) + \left(\frac{k^2(z)}{\rho(z)} - \frac{\lambda}{\rho(z)}\right)u(z) = 0 \tag{1.14}$$

where λ is the separation constant, $u(z)$:[0,∞) with:

$$u(z) = \begin{cases} u^{(1)}(z), & \text{if } 0 \leq z \leq D \\ u^{(2)}(z), & \text{if } D < z \leq D + h \\ u^{(3)}(z), & \text{if } z > D + h \end{cases}$$

and

$$k(z) = \begin{cases} k_1(z) = \frac{\omega}{c_1(z)}, & \text{if } 0 \leq z \leq D \\ k_2(z) = \frac{\omega}{c_2(z)}, & \text{if } D < z \leq D + h \\ k_3(z) = \frac{\omega}{c_3}, & \text{if } z > D + h \end{cases}$$

The functions $u(z)$ are called *eigenfunctions* or *modes*, λ the *eigenvalues* and $k(z)$ is the *wave number*. Note that we have to assume that the velocity in the seabed is constant in terms of depth, because the domain is semi-infinite.

The equation (1.14) describes a *singular* Sturm-Liouville problem, defined in the infinite domain. Combining (1.14) with the following conditions, we define the *depth problem*:

$$u^{(1)}(0) = 0 \tag{1.15}$$

$$\lim_{z\to D^-} u^{(1)}(z) = \lim_{z\to D^+} u^{(2)}(z) \tag{1.16}$$

$$\lim_{z\to D^-} \frac{1}{\rho_1}\frac{\partial u^{(1)}(z)}{\partial z} = \lim_{z\to D^+} \frac{1}{\rho_2}\frac{\partial u^{(2)}(z)}{\partial z} \tag{1.17}$$

$$\lim_{z\to (D+h)^-} u^{(2)}(z) = \lim_{z\to (D+h)^+} u^{(3)}(z) \tag{1.18}$$

$$\lim_{z\to (D+h)^-} \frac{1}{\rho_2}\frac{\partial u^{(2)}(z)}{\partial z} = \lim_{z\to (D+h)^+} \frac{1}{\rho_3}\frac{\partial u^{(3)}(z)}{\partial z} \tag{1.19}$$

$$\lim_{z\to \infty} u^{(3)}(z) = 0 \tag{1.20}$$

We state the following theorem for a *singular* Sturm-Liouville problem:

Theorem 1. *(Representation Theorem) Let* $\{u_n(z)\}_{n=1}^N$, $\{u(z,\lambda),\ \lambda\in S\ \}$ *the set of eigenfunctions that correspond to the discreet and continuous eigenvalues spectrum respectively , of a Sturm-Liouville problem defined in an interval [a,*∞*).Then every continuous function f that satisfy the boundary conditions of this problem and has piecewise continuous derivative in the interval [a,*∞*), is approximated from:*

$$f(z) = \sum_{n=1}^{N} A_n u_n(z) + \int_S b(\lambda)u(z,\lambda)d\lambda \tag{1.21}$$

where $A_n\in\mathbb{R}$*, n=1,...,N and b:S*$\to\mathbb{R}$ *a function.*

In our case, according to (1), we can describe pressure through the formula:

$$p(r,z) = \sum_{n=1}^{N} A_n(r)u_n(z) + \int_S b(r;\lambda)u(z,\lambda)d\lambda \tag{1.22}$$

since it is a continuous function of z in [0,∞), has piecewise continuous derivative and satisfies the boundary conditions. It is important to mention that the second term refers to the energy loss in the seabed and does not contribute to the acoustic field away from the source. So we can omit this term:

$$p(r,z) \approx \sum_{n=1}^{N} A_n(r)u_n(z), \quad r \gg \frac{1}{k(z)} \tag{1.23}$$

Each term in the sum $\{A_n(r)u_n(z)\}_{n=1}^N$ is called *mode*. The equation (1.14) can be written as a system of equations:

$$\frac{d^2u_n^{(*)}(z)}{dz^2} + (k^2(z) - \lambda)u_n^{(*)}(z) = 0 \tag{1.24}$$

One can see that, the equation (*)=3 can be solved analytically. That does not hold,however , for the cases (*) = 1,2 ,since the *wave number* (and so the sound velocity) depends on depth. Nevertheless, this does not prevent us to expand the pressure with normal mode series. To find the eigenvalues, we should solve an equation in the form $f(\lambda) = 0$ (*characteristic equation of the waveguide*)numerically, taking into consideration that the eigenvalues must satisfy:

$$\frac{\omega^2}{c_3^2} < \lambda_n < \frac{\omega^2}{\min\{c_{1,\min}^2, c_{2,\min}^2\}} \tag{1.25}$$

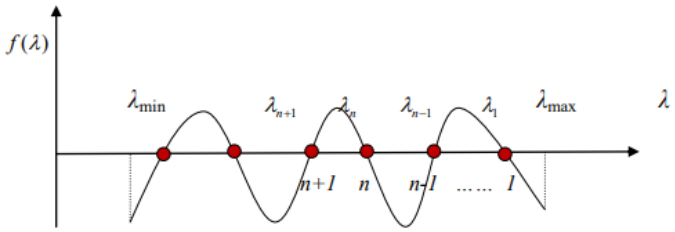


Figure 1.2: Possible sketch of f(λ) (obtained from [17])

1.3.2 Representation of the solution

As we saw, due to the fact that the domain is not bounded for large depths, calculations show that the discreet spectrum constitutes only finite number of eigenvalues (and thus modes, which are called *normal modes*).Reminding that we want to calculate the pressure far away from the source, we write:

$$p(r,z) = \sum_{n=1}^{N} A_n(r) u_n(z) \tag{1.26}$$

Substituting (1.26) to (1.5) and using the properties of Dirac and orthogonality of modes, we get:

$$\frac{d^2 A_n(r)}{dr^2} + \frac{1}{r}\frac{dA_n(r)}{dr} + \lambda_n A_n(r) = -\frac{1}{2\pi r}\frac{1}{\rho(z_0)}\delta(r) u_n(z_0) \tag{1.27}$$

Since the source is set to be in the water (first layer), $\rho(z_0) = \rho_1$ and $u_n(z_0) = u_n^{(1)}(z_0)$ and so (1.27) can be rewritten as:

$$\frac{d^2}{dr^2}\left[\frac{\rho_1}{u_n^{(1)}(z_0)} A_n(r)\right] + \frac{1}{r}\frac{d}{dr}\left[\frac{\rho_1}{u_n^{(1)}(z_0)} A_n(r)\right] + \lambda_n \left[\frac{\rho_1}{u_n^{(1)}(z_0)} A_n(r)\right] = -\frac{1}{2\pi r}\delta(r) \tag{1.28}$$

The function $\frac{\rho_1}{u_n^{(1)}(z_0)} A_n(r)$ is the *Green function* of (1.28) and it can be shown that is equal to:

$$\frac{\rho_1}{u_n^{(1)}(z_0)} A_n(r) = \frac{i}{4} H_0^{(1)}(\sqrt{\lambda_n} r) \tag{1.29}$$

where $H_0^{(1)}(\sqrt{\lambda_n} r)$ is the *Hankel function* of the first kind and zero order. We use the asymptotic form of *Hankel function* $H_0^{(1)}(r)$ for long distances (r $\gg$ 1) which is given by the formula:

$$H_0^{(1)}(r) \sim \sqrt{\frac{2}{\pi r}} e^{i\left(r - \frac{\pi}{4}\right)}, \quad r \gg 1 \tag{1.30}$$

So, (1.26) can be written as:

$$p(r,z) = \frac{i}{4\rho_1} \sum_{n=1}^{N} H_0^{(1)}(\sqrt{\lambda_n} r) u_n^{(1)}(z_0) u_n(z) \tag{1.31}$$

or equivalently

$$p(r,z) = \frac{i}{4\rho_1} \sum_{n=1}^{N} u_n^{(1)}(z_0) u_n(z) \sqrt{\frac{2}{\pi\sqrt{\lambda_n} r}} e^{i\left(\sqrt{\lambda_n} r - \pi/4\right)} [17] \tag{1.32}$$

Note that the computation of the *eigenfunctions* is, usually, done numerically unless the sound velocity is constant and we have analytical solution.

It is important to mention that, in reality the pressure does not only depend on r and z.In fact, pressure depends also on source depth z_0 and angular frequency ω which connects directly with the *eigenvalues* through the module of the *wavenumber*. That is pressure on the frequency domain can be written as: $p = p(r, z; z_0; \omega)$[6]

1.4 Broadband source

1.4.1 Broadband source and its acoustic field

Now, we are going to describe another case of acoustic excitation. Suppose, that it emits a signal consisting of different frequencies[18]. Thus we will consider a source excitation function of the form $S(\omega)$. In our work we will consider a source with a Gaussian distribution in the frequency domain:

$$S(\omega) = e^{-\frac{\pi(\omega-\omega_0)}{(\Delta\omega)^2}} \tag{1.33}$$

where ω_0 is the *central frequency* and $\Delta\omega$ is the *semi-width* of the distribution.

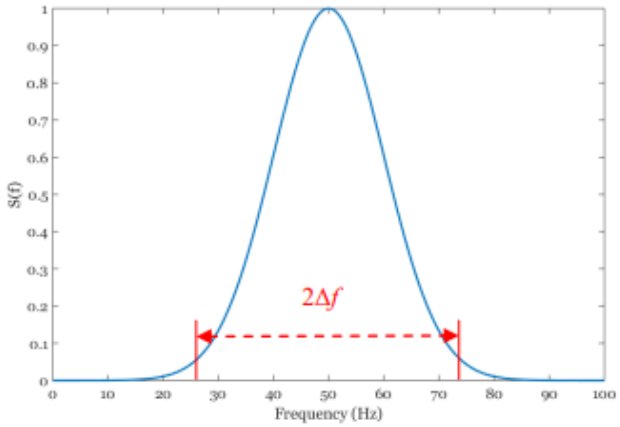


Figure 1.3: An example of Guassian Source (obtained from [17])

How can we calculate acoustic pressure that is created from this kind of source? We use the expression of $p(r, z; z_0, \omega)$ that is created from a harmonic source. Namely, we calculate the acoustic field $p(r, z; z_0, \omega)$ across all the frequencies emitted from the source by uniformly discretizing the frequency spectrum $[\omega_0 - \frac{\Delta\omega}{2}, \omega_0 + \frac{\Delta\omega}{2}]$. When we calculate the acoustic field, we assume that we have an harmonic source with unit amplitude for every discreetised frequency ω_i in the previous interval. Then, we multiply the result with the source distribution on the corresponding frequency ω_i, based on filter theory.[3]. Summing up, the acoustic field produced by a source that follows a Gaussian distribution in the frequency field is:

$$\mathcal{P}(r, z, z_0; \omega) = p(r, z, z_0; \omega) S(\omega) \tag{1.34}$$

If one would like to find the function $s(t)$, which describes the signal in the *time domain*, can use *inverse discreet Fourier transform*:

$$s(t) = \mathcal{F}^{-1}\{\mathcal{P}(r, z, z_0; \omega)\} \tag{1.35}$$

which is will give us:

$$s(t) = \mathcal{P}(r, z, z_0; t) \tag{1.36}$$

1.4.2 Group Velocity

We shall introduce the context of *group velocity*. When energy is transferred from range of frequencies and the acoustic signal is not harmonic, then every *mode* has its *own* rate of energy transfer. As a result, one can observe mode separation with time, as a signal propagates in the *waveguide*. We are going to define the *group velocity* which is unique for every mode and represents the variation of *angular frequency* with respect to the horizontal direction of propagation and also the energy transport velocity

[3] See *Convolution theorem*

of a particular mode in the waveguide . Suppose that we have a signal in the frequency domain $s(\omega)$ caused by a narrow band-source , then in time domain the signal carried by the n^{th} mode is:

$$s(t) = \int_{\omega-\epsilon}^{\omega+\epsilon} s(\omega)\, e^{-i[\omega t - \kappa_n(\omega) r]} d\omega \tag{1.37}$$

For a small time the wave will propagate at horizontal distance $dr = v_n dt$. For the signal to be unchanged over the time interval dt all components of the integral must stay in phase. Which means that $d\omega\, dt - d\kappa_n(\omega)\, dr = 0$. Combining the previous two equations we get:

$$v_n(\omega) = \frac{d\omega}{d\kappa_n}^{4} \tag{1.38}$$

where n denotes the mode number.

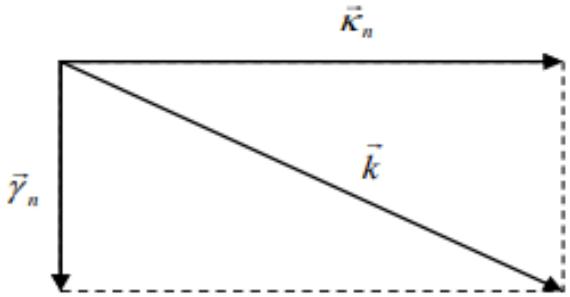


Figure 1.4: Wavenumber and its components (obtained from [17])

A method that approximates the group velocity well is based on *perturbation theory*[6]:

$$\frac{1}{v_n(\omega_0)} = \left.\frac{d\kappa_n}{d\omega}\right|_{\omega_0} = \frac{\omega}{\kappa_{n,0}} \int_0^{\mathcal{Z}} \frac{u_{n,0}^2(z)}{\rho(z)c^2(z)} dz \tag{1.39}$$

where is the lower bound of the *waveguide* boundary (in our case it is ∞), $u_{n,0}(z)$ is the n^{th} class *eigenfunction* calculated at frequency ω_0 and $\kappa_{n,0}$ is the horizontal component of the wave number calculated at frequency ω_0. Note, that this approximation works really well when the source emits a narrow band signal. That is, when all frequencies are close to the central frequency ω_0, which represents the velocity of energy propagation of the signal in the *waveguide*. Another approximation, which was used in our problem, is to calculate *group velocities* through derivative approximation, namely:

$$v_n = \frac{(\omega + \Delta\omega) - \omega}{\kappa_n(\omega + \Delta\omega) - \kappa_n(\omega)} \tag{1.40}$$

Note, that one has to choose $\Delta\omega$ carefully, because large $\Delta\omega$ gives inaccurate results and using small $\Delta\omega$ round-off error can corrupt the answer. Lastly, we note that we should not confuse *group velocity* with the concept of *phase velocity*, which represents the horizontal velocity of a particular phase in the plane-wave representation of a mode and is given by the formula $v_{p,n} = \frac{\omega}{\kappa_n}$

[4] Check that $|\vec{\kappa_n}| = \sqrt{\lambda_n}$ from equation (1.24)

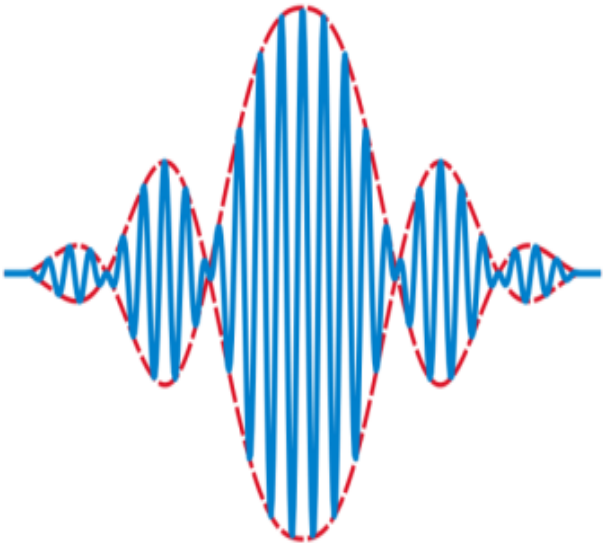


Figure 1.5: Wave propagation in time domain. The shape that is noted with red lines, propagates with *Group Velocity* (obtained from [17])

CHAPTER 2

Signal Processing and Analysis

In the previous chapter we formulated *normal mode theory* using monochromatic signals. The insertion of a *broadband source* in our theory, adds the need to use *Fourier transform.* Firstly, we will introduce the types of *Fourier transforms* that are going to be used:

- Fast Fourier Transform (FFT)
- Short Time Fourier Transform (STFT)

2.1 Fast Fourier Transform

Discrete Time-Frequency Fourier Transform (DFT)pair $F[k] \longleftrightarrow f[n]$ is given by :

$$F[k] = \frac{1}{N} \sum_{n=0}^{N-1} f[\,n]\, e^{-ink(2\pi/N)} \tag{2.1}$$

and the inverse :

$$f[n] = \sum_{k=0}^{N-1} F[\,k]\, e^{ink(2\pi/N)} \tag{2.2}$$

where T is the transform period and N is the number of samples of the given signal[10]. In the context of Fast Fourier Transform we symbolize $w_N = e^{i(2\pi/N)}$ and we simplify the notation:

$$F[k] = \frac{1}{N} \sum_{n=0}^{N-1} f[\,n]\, w_N^{-nk} \tag{2.3}$$

$$f[n] = \sum_{k=0}^{N-1} F[\,k]\, w_N^{nk} \tag{2.4}$$

The reason we use FFT is that reduces the complexity of computing the DFT from $O\left(N^2\right)$ to $O(N \log_2 N)$. It is a smart way to compute the DFT and the difference in speed is significant, especially for a long sampled signal.Typically, to utilize the speed of FFT algorithm we manually choose the sample number N to be a power of 2 (1024, 2048, 4096 etc). The interested reader can see in details the algorithm of FFT in the bibliography.[3] Briefly, the FFT can be described as following: Let the array $[F[0], F[2], ..., F[N-1]]^T$

be the Fourier coefficients and $[f[0], f[2], ..., f[N-1]]^T$ the sampled signal, the equation (2.3) can be written as matrix multiplication:[14]

$$\begin{bmatrix} F[0] \\ F[2] \\ \vdots \\ \vdots \\ F[N-1] \end{bmatrix} = \frac{1}{N} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & w_N^1 & w_N^2 & \cdots & w_N^{N-1} \\ 1 & w_N^2 & w_N^4 & \cdots & w_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w_N^{N-1} & w_N^{2(N-1)} & \cdots & w_N^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} f[0] \\ f[2] \\ \vdots \\ \vdots \\ f[N-1] \end{bmatrix} \tag{2.5}$$

or equivalently:

$$\begin{bmatrix} F[0] \\ F[2] \\ \vdots \\ \vdots \\ F[N-1] \end{bmatrix} = \frac{1}{\sqrt{N}} \boldsymbol{W}_N \begin{bmatrix} f[0] \\ f[2] \\ \vdots \\ \vdots \\ f[N-1] \end{bmatrix} \tag{2.6}$$

where the $N \times N$ matrix $\boldsymbol{W}_N = \frac{1}{\sqrt{N}}[w^{(k-1)(j-1)}]_{1 \le i,j \le N}$ is the *Fourier matrix* . Executing the matrix multiplications, we would need N^2. By rearranging the elements of the matrices we can achieve the desired result only with $Nlog_2N$ multiplications. Namely we can decompose the previous matrix multiplication as:

$$\boldsymbol{F} = \frac{1}{\sqrt{N}} \boldsymbol{W}_N \boldsymbol{f} = \frac{1}{N} \begin{bmatrix} \boldsymbol{I}_{N/2} & \boldsymbol{D}_{N/2} \\ \boldsymbol{I}_{N/2} & -\boldsymbol{D}_{N/2} \end{bmatrix} \begin{bmatrix} \boldsymbol{W}_{N/2} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{W}_{N/2} \end{bmatrix} \begin{bmatrix} odd-even \\ permutation \end{bmatrix} \boldsymbol{f} \tag{2.7}$$

$$= \begin{bmatrix} \boldsymbol{I}_{N/2} & \boldsymbol{D}_{N/2} \\ \boldsymbol{I}_{N/2} & -\boldsymbol{D}_{N/2} \end{bmatrix} \begin{bmatrix} \boldsymbol{W}_{N/2} & \boldsymbol{O} \\ \boldsymbol{O} & \boldsymbol{W}_{N/2} \end{bmatrix} \begin{bmatrix} \boldsymbol{f}[\text{even}] \\ \boldsymbol{f}[\text{odd}] \end{bmatrix}$$

where **I** is the identity matrix and $\boldsymbol{D}$ a diagonal matrix. The index $N/2$ of the top matrix $\boldsymbol{D}_{N/2}$ denotes that it contains the elements $w_N^1, ..., w_N^{N/2}$. Matrix $\boldsymbol{W}_{N/2}$ contains the elements of a *Fourier matrix* given by $N/2$ samples. The permutation matrix groups the elements of array $\boldsymbol{f}$ into elements with even and odd index. By doing this decomposition repeatedly, we can keep reducing the dimensions of the $\boldsymbol{W}$ blocks up to 2×2 matrices, simply by rearranging the odd and even indexed elements of the sampled function in each step. We notice that, each time the dimension is reduced by a factor of 2. This is why we like to choose the number of samples to be a power of 2, when we want to use FFT.

It is also important to note, that one has to be careful when chooses the number of samples, given the maximum frequency of the bandwidth. Not every number (that is a power of two) is appropriate for. This happens because, the *sample frequency* determines if we can retrieve the signal in the time domain after using the Fourier transform. We can choose N using the following theorem:

Theorem 2. *(Sampling Theorem) A bandlimited signal with $F(\omega)$[1] for $|\omega| > \omega_{max}$ can be fully represented by its samples, provided the sampling frequency ω_s equals or exceeds twice the highest signal frequency ω_{max}, that is $\omega_s > 2\omega_{max}$.*

Applying the theorem we have:

$$\frac{\omega_s}{2} > \omega_{\text{max}} \iff \frac{Nd\omega}{2} > \omega_{max} \iff N > 2\frac{\omega_{max}}{d\omega} \tag{2.8}$$

where $d\omega$ is the *step frequency* of the frequency domain partition. The time interval $T = \frac{2\pi}{\omega_s} = \frac{\pi}{\omega_{\text{max}}}$ is called the *Nyquist interval* and $2\omega_{\text{max}}$ is the *Nyquist frequency*. In other words, theorem (2) indicate how fast we should do our sampling. If we do not sample as the theorem says, then when we return to the time

[1] In our problem, the $F(\omega)$ of theorem 2 is $S(\omega)$

domain we will get an artifact of the original signal. This phenomenon is called *aliasing*. When we do not follow the inequality of the theorem, we say the we have *undersampled* the signal. *Aliasing* can be noticed also in the frequency domain, where the graph that appears periodically with frequency ω_s, overlaps with the next and the previous ones appeared.

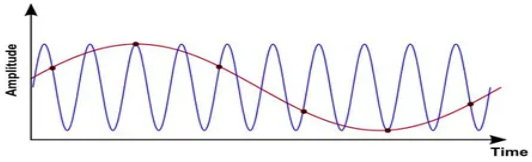


Figure 2.1: Aliasing Effect: The original signal is the blue curve and the dots are the sample that we have taken. The red signal is the result of the aliasing effect. (obtained from the World Wide Web)

2.2 Short Time Fourier Transform

The use of FFT, and in general of DFT, is significant in many applications, however a main drawback is that it provides only one picture that "averages" the frequency components across the whole duration of the signal. We know what frequency components are present but we do not know when. To gain the advantage of knowing the frequency in addition with time domain, we use STFT (*Gabor transform*)[11]. The idea is to consider small segments of the signal ,which are called *frames* ,and then we apply a DFT for **each** frame. One can visualize the STFT process from the following sketch:

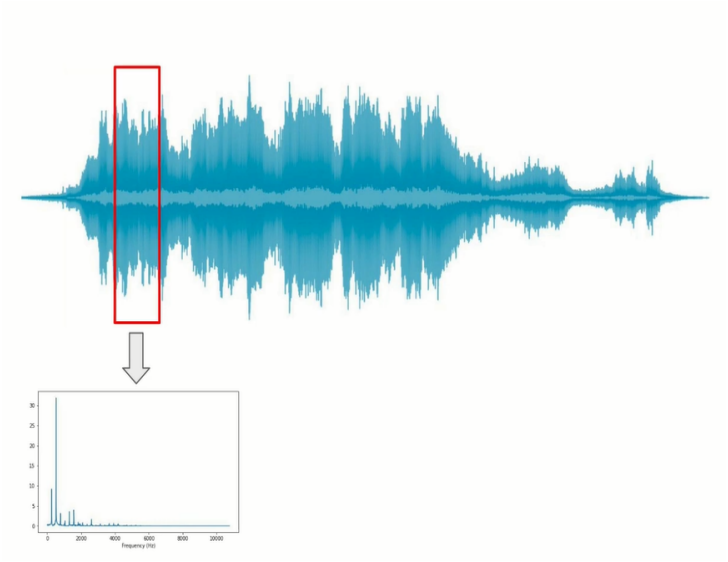


Figure 2.2: Visualizing STFT (obtained from the World Wide Web)

To derive the STFT coefficients we use *windowing*. In other words, we apply a *window function* to a signal. The signal after this process is symbolized as $f_{g[n]} = f[n]g[n]$, where $g[n]$ is the window function and $f[n]$ the original signal. A typical example of a window function is *Hann window* with the following formula: $g[n] = 0.5 - 0.5\cos(\frac{2\pi n}{M-1}), \quad n = 0, ..., M$ where $M \leq N$ is the $window\ size$.

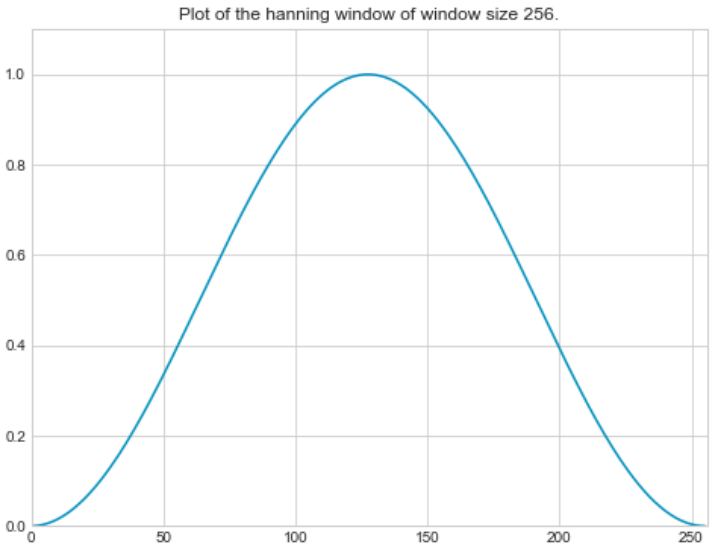


Figure 2.3: Hann window with M = 256 (obtained from the World Wide Web)

As we can see the *window size* is the number of samples we apply windowing to. Nevertheless, we should consider another parameter with called *frame size*, which is the amount of samples that we consider when we segment the signal for the FT. Usually those quantities are the same, but if we decide for example that *frame size* is greater than *window size* what happens is that we apply the FT to the whole frame and the windowing occurs only on the window size number of samples. The remaining samples, which are the difference between the *frame size* and the *window size*, are going to give zero contribution. Another parameter that we use in STFT is *hop size*, which is the amount of samples we slice to the right, when we change frame. So, in many cases, *hop size* leads to overlapping frames which prevents spectral leakage due to possible discontinuities that are caused from the windowing.

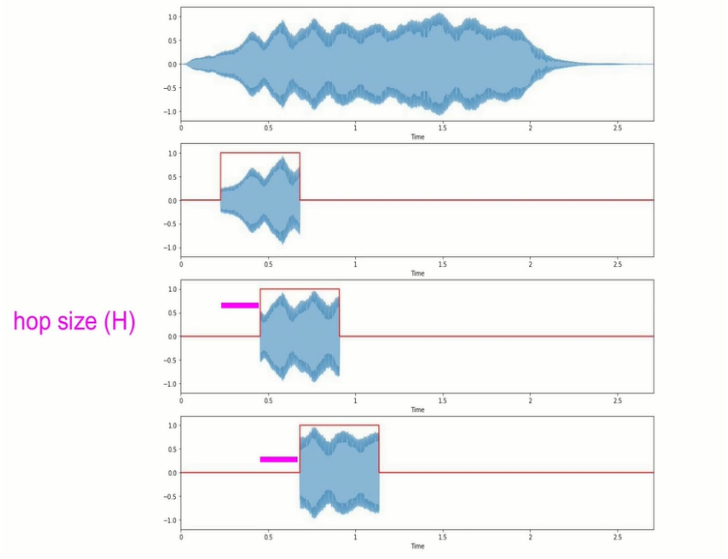


Figure 2.4: Visualization of Hop Size (obtained from the World Wide Web)

The mathematical formulation for STFT is:

$$S\left[m,k\right]=\sum_{n=0}^{L-1}f\left[n\right]g\left[n-mH\right]e^{-ink(2\pi/L)} \tag{2.9}$$

which represents the Fourier coefficient for the k^{th} frequency at the m^{th} frame of length $L<N$ using

hop size H. Note that *frame size*, *window size* and *hop size* are powers of two.

It holds: $\omega_{min} \leq \omega^{(k)} \leq \omega_{max}$, where the interval $[\ \omega_{min}, \omega_{max}\ ]$ is partitioned using step frequency $d\omega$: $\omega^{(i+1)} = \omega^{(i)} + d\omega$.

It is important to mention the Time-Frequency trade of, namely to find the golden intersection between frequency and time resolution we have to lower or increase frame size. This is not an easy task, because by increasing *frame size* we increase frequency and lower time resolution and vice versa.

2.3 Spectrogram

The main reason applying SFT is to **visualize sound**, which as we will see, plays significant role in signal processing. The visualization of sound is given by the *spectrogram* (time-frequency domain representation) that uses :

$$Y\,[m,k] = |S\,[m,k]\,|^2 \tag{2.10}$$

which is a matrix with dimensions *frequency bins*×*frames* [2] and components the squared module of the corresponding m^{th} and k^{th} Fourier coefficient. It describes the way the energy levels vary in a specific time stamp and denotes the energy density of $f\,[n]$ in a time-frequency neighbourhood.The visualization is accomplished through a heat-up process.

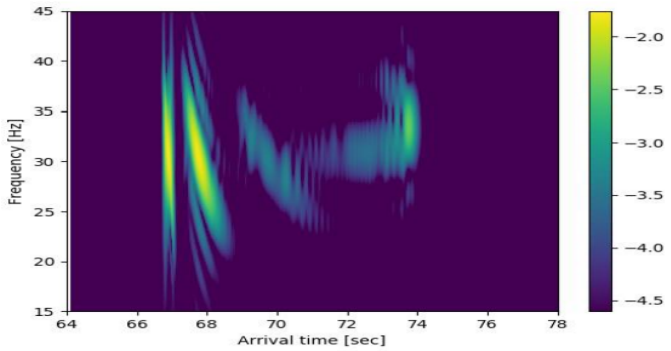


Figure 2.5: Spectrogram example on logarithmic scale

2.4 Dispersion Curves

Dispersion curves describe the *acoustic dispersion* phenomenon, that is the procedure when a sound wave, which is emitted from a broadband source, separates into wave packets that propagate through a material (in this case the waveguide), at different velocities (*group velocities*). They describe the non-linear relation of the angular frequency with the wavenumber of the mode n, with slope equal to the *group velocity*[13]. That is, the separation into wave packets, is measured by the rate of change of the *group velocity*, which as we mentioned before is a function of frequency. The non-linear relationship is given by the *characteristic equation* of the waveguide if we solve in terms of ω. One dispersion curve can be plotted by calculating the *group velocity* and the corresponding *arrival time* that is given from the formula $\frac{R}{v_n}$, where R is the receiver range from the source and v_n is the group velocity on the n^{th} *eigenvalue*. An important observation is that the cut-off frequency, meaning the frequency that is the lowest possible that allows propagation phenomenon, can be found by searching the point where the *dispersion curve* stops. As we study higher order of modes, we observe higher cut-off frequency bounds, that is higher order modes arrive later. Another important observation is that in many cases, the *dispersion curves* have a returning behaviour[6], due to the fact that the group velocity has local extrema at a certain frequencies,

[2] *frequency bins* = $\frac{framesize}{2}$+1 and *frames* = $\frac{samples - framesize}{hopsize} + 1$

which in time-domain solutions give rise to the so-called *Airy phase* forming the tail of a transient modal arrival.

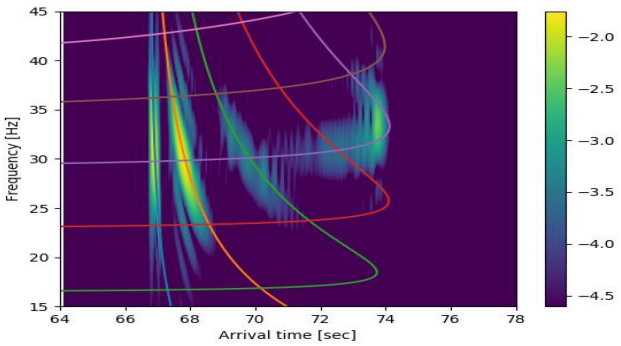


Figure 2.6: Spectrogram example on logarithmic scale with its Dispersion Curves

The details about the characteristics of the environment of the example above, are shown in the following table[3]:

Parameters describing the marine environment
Water depth (**D**) **200 m**
Sediment thickness (**h**) **30 m**
Source range (**R**) **100000 m**
Central frequency (f_c) **30 Hz**
Bandwidth frequency (Δf) **30 Hz**
Source/receiver depth ($\mathbf{z_0}$, $\mathbf{z_r}$) **60 m**
Sound velocity profile in the water ($\mathbf{c_w}(z)$) $\mathbf{c_w}(0)$ **1505 m/sec** $\mathbf{c_w}(40)$ **1500 m/sec** $\mathbf{c_w}(60)$ **1497 m/sec** $\mathbf{c_w}(150)$ **1507 m/sec** $\mathbf{c_w}(200)$ **1510 m/sec**
Water density (ρ_w) **1000 kg/m^3**
Sound velocity profile in the sediment ($\mathbf{c_b}(z)$) $\mathbf{c_b}(200)$ **1650 m/sec** $\mathbf{c_b}(230)$ **1700 m/sec**
Sediment density (ρ_b) **1700 kg/m^3**
Sound velocity in the seabed ($\mathbf{c_b}$) **1800 m/sec**
Seabed density ($\rho_{\mathbf{sb}}$) **2000 kg/m^3**

Table 2.1: Describing the Environment of the example

[3]The spectrograms and the parameters of the example was taken from the thesis of Savinas Antoulinaki : Statistical characterization of signals using wavelet packets

CHAPTER 3

Inverse Problem Theory

3.1 Forward and Inverse Problem

In mathematical modeling, we call a problem *forward* when we are asked to determine quantities, usually in the form of a function, when the parameters and the equation or equations that condition the problem are known. On the other hand in an *inverse problem*, we are asked to determine the parameters of a physical phenomenon or mathematical problem when the *measurements*, which might describe the values of a function, are known. The measurements, in the inverse problem terminology, are called *data* and the unknowns, as we saw, *model parameters*. Both of them, can be continuous quantities or discrete. In the latter case, we use the term: *discrete inverse problem*. In our project, we are focused on a discrete inverse problem.

Forward Problem:

$$\text{Model Parameters} \longrightarrow \text{Model} \longrightarrow \text{Prediction of Data}$$

Inverse Problem:

$$\text{Data} \longrightarrow \text{Model} \longrightarrow \text{Estimates of Model Parameters}$$

3.2 Discrete Inverse Problem Formulation

Assuming that both *data* and *model parameters* take discrete values, we define the vector $\mathbf{d}=[d_1, d_2, ..., d_N]^T$ that represents all the measurements and $\mathbf{m}=[m_1, m_2, ..., m_M]^T$ that represents all the parameters to be found. In the general case, the formula that connects $\mathbf{d}$ with $\mathbf{m}$ is:

$$f_j(\mathbf{d}, \mathbf{m}) = 0, \quad j = 1, ..., L \tag{3.1}$$

or more compactly:

$$\mathbf{f}(\mathbf{d}, \mathbf{m}) = \mathbf{0} \tag{3.2}$$

We have to add that, in many cases $\mathbf{d}$ and $\mathbf{m}$ might be represented as random variables, due to uncertainties that may arise from the nature of the problem.
In the following subsections we will name some frequent cases of inverse problems:

3.2.1 Implicit Linear Form

The function **f** is linear in both data and model parameters and thus can be written the form of a $L \times (M + N)$ matrix:

$$\mathbf{f}(\mathbf{d}, \mathbf{m}) = \mathbf{0} = \mathbf{F} \begin{bmatrix} \mathbf{d} \\ \mathbf{m} \end{bmatrix} \tag{3.3}$$

however they are not separable.

3.2.2 Explicit Form

The function **f** is linear only in respect of data and thus we can separate **d** and **m**. There are $L = N$ equations:

$$\mathbf{f}(\mathbf{d}, \mathbf{m}) = \mathbf{0} = \mathbf{d} - \mathbf{g}(\mathbf{m}) \tag{3.4}$$

3.2.3 Explicit Linear Form

In this case data and model parameters are separable and function **g** is linear. As above, $L = N$ and also we define a matrix **G** with dimensions $N \times M$:

$$\mathbf{f}(\mathbf{d}, \mathbf{m}) = \mathbf{0} = \mathbf{d} - \mathbf{G}\mathbf{m} \tag{3.5}$$

In every previous case, one can not be sure that there exists a unique solution **m**. In fact there could be zero to infinite solutions, however since our interests are in physical applications we know that solutions exist. Our goal in that kind of problems is to find appropriate approximations of the model parameters **m**.

Note, that since model parameters are going to be approximated, depending on the way of processing the problem that is set, would be wise to consider them as *random variables*. Furthermore, in most studies ,where measurements are involved, there are errors in the measurements for multiple reasons such as inaccurate tools. So, in these cases on could also assume that the known vector **d** is a *random variable*. It is clear that in inverse problems there many ways of approaching them and usually the correct methodology is determined by trial-error technique.

3.3 Nonlinear Inverse Problems

In the previous section, we indirectly covered some of a vast amount of inverse problems. The *nonlinear* inverse problems. Namely, they could have the form $\mathbf{d} - \mathbf{g}(\mathbf{m}) = \mathbf{0}$, $\mathbf{f}(\mathbf{d}, \mathbf{m}) = \mathbf{0}$. Nevertheless, the problem that we are trying to formulate, may not be possible to be written in such a closed form, as we will see in the description of the inverse problem that we are trying to solve.

3.3.1 Parameterizations

In any inverse problem, especially *nonlinear*, it is important to choose the appropriate variables to represent our data (*measurements*), parameters and the model that connects the previous two. An appropriate *parameterization* will not only satisfactorily connect **d** and **m**, but will also simplify the solution procedure. An illustration of the importance of the parameterization is the following: consider the problem of fitting a line with data points (1,1), (2,2), (3,3), (4,5). Then, considering the pairs (z, d) we get the line $d = -0.5 + 1.3z$. whereas if we regard them as (d', z') then least squares give $d' = 0.309 + 0.743z' \Longleftrightarrow z' = -0.416 + 1.345d'$. These two line have slopes (m_1) and intercept (m_2) that differ by $\sim$20%. One should also be cautious on the usage of change of variables in order to solve the problem more easily. In general, we might not want find the parameters themselves, but their estimates[1] for example *mean value* or *maximum likelihood point*. Estimates are **not** invariant under changes of the

[1] Here we suppose that **m** are random variables

parameterization. So, if $\mathbf{m}$ and $\mathbf{m}'$ are two different parameterizations we want to avoid a procedure like: distribution for $\mathbf{m} \rightarrow$ estimate of $\mathbf{m} \rightarrow$ estimate of $\mathbf{m}'$ and prefer: distribution for $\mathbf{m} \rightarrow$ distribution of $\mathbf{m}' \rightarrow$ estimate of $\mathbf{m}'$, which however usually involves more complicated mathematics.

3.3.2 Inverse Problems and Optimization

The main goal, when one encounters a *nonlinear* inverse problem is to find the minimum (or maximum) of an *objective function*. That is, we use *optimization* methods to solve the inverse problem. For example, one might need to find the maximum likelihood point of a distribution $P(\mathbf{x})$ and as result try optimize $P(\mathbf{x})$. In the next sections we add more information about optimization theory. We have to note that in some cases, although it does not always work, it is sufficient to linearize the model. As we mentioned above, this should be done with caution, because linearizing frequently means changing parameterization. Another way to linearize the problem is to take the Taylor expansion of the model and ignore terms of second order and higher.

3.3.3 Convergence and Uniqueness

In *nonlinear* inverse problems there is **no** simple way to determine if they have a unique solution that, as we mentioned before, it minimizes (or maximizes) an *objective function*. In many cases, it has more than one minima (or maxima) which not only does not imply uniqueness, but it also might slow or eliminate the convergence of our method. In some problems when we start with an initial value near the extremum can guarantee convergence[9]. In practice the main problem one has to deal with is not the existence of multiple extrema, but the plethora of saddle points that occur due to the wave-like form of the *objective function*. This occurs to a great extent when the dimension of the unknown parameters vector is high. In this case, one should implement more clever optimization methods, we cite some of them in the appendix.

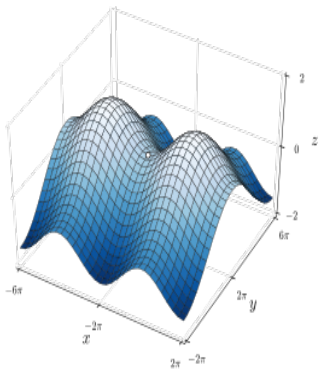


Figure 3.1: Objective Function example (obtained from the World Wide Web)

3.4 Inverse Problems in Underwater Acoustics

This work is associated with a class of inverse problems in underwater acoustics which use as input data, characteristics of an acoustic signal emitted from a known source to recover parameters of the marine environment. These parameters are typically the sound speed profile in the water column and/or the geoacoustic parameters of the sea-bed. The model associating the features of the acoustic signal with the environmental parameters has been implicitly presented in the first chapter of this work.

3.4.1 Inverse Problem: Using dispersion characteristics

Among the different approaches presented in literature exploiting the measurements to define the data to be used in the formulation of the inverse problem, we will consider those using a single hydrophone as the reception device and exploiting the mode dispersion quantified through the dispersion curves of a

spectrogram. The dispersion characteristics are unique features of the signal as it has propagated through the environment and therefore they are directly associated with the environmental parameters.The inverse problem thus defined is non-linear. In order that the dispersion characteristics are used as input data to the inverse problem, they have to be identified in the measured signal. The identification is based on the spectrogram and several methods have been reported for the definition of the input data to the inverse problem, which in most cases are time and frequency values of the dispersion curve associated with each propagating mode. This association requires that the dispersion curves are recognized in the measured signal. In summary, the inverse problem can be described by the following statement : Given the spectrogram of an acoustic signal recorded in the marine environment into consideration, identify the *dispersion curves* that describe the energy distribution of the measured signal, and use their mathematical description in the form of discrete data to estimate the environmental parameters. Due to the nature of the problem it is solved by some optimization process.

3.4.2 Describing dispersion curves with splines

Our work is devoted to the identification of the dispersion characteristics of the measured signal based on its spectrogram. We will describe the dispersion curves with *cubic splines*. Thus, our final goal is the estimation of the coefficients of the splines representing the dispersion curves of the spectrogram of the measured signal. These coefficients not only determine the shape of the dispersion curves, but could also be used as discrete input data in the framework of an inverse problem as described in the previous sub-section. We will make the reasonable assumption that we have some a-priori information on the environmental parameters to be recovered and we will use *Convolutional Neural Networks* (CNN) appropriately trained, with synthetic spectrograms and associated dispersion curves, to provide as their output the spline coefficients of the dispersion curves of the measured signal. Synthetic signals will be calculated using code mode1[2] developed by Michael Taroudakis[15] which provides the acoustic field in the frequency domain for a known acoustic source. The estimation of the coefficients of the dispersion curves of the measured signal constitute an inversion procedure to be sketched below.

3.4.3 Summary of the Inverse Problem Formulation

The following graph illustrates the procedure of estimating the spline coefficients from the spectrogram:

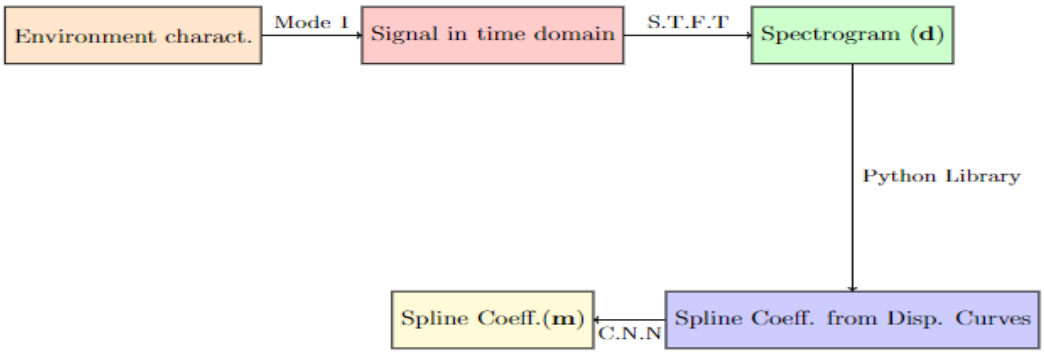


Figure 3.2: Sketch of the Solution

Note, that CNN represent the model $\mathbf{f}(\mathbf{d}, \mathbf{m}) = \mathbf{0}$

[2] Since the signals are synthetic and not real, we chose to not add noise

CHAPTER 4

Introduction to Neural Networks

In this chapter, we will briefly cover some fundamental ideas of *Neural Networks* (NN) whose key aspect is that their internal mechanism is not designed by the user: they are learned from data using a general procedure based on non-linearity. This part of the theory is important for one to understand the way of solving our problem. In the appendix we will provide some additional references on NN for the interested reader.

4.1 Neural Networks

The term *Neural Network* surrounds a great variety of methods and techniques. Usually, one can try to interpret the structure of the N.N, as the way the neurons of the human brain function. However, it is useful for the reader to have a more compact understanding of their formulation. A very simple NN can be seen in the following figure:

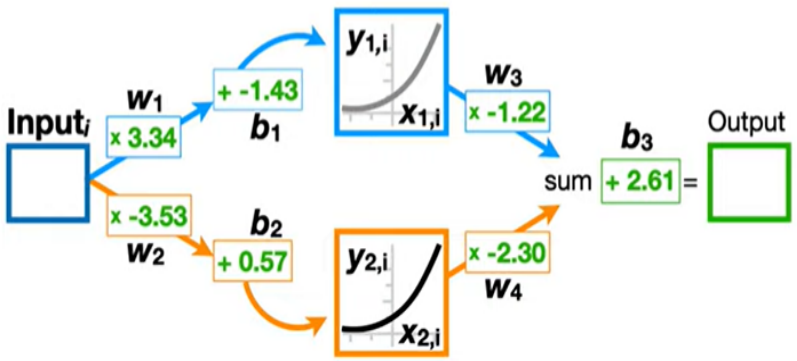


Figure 4.1: A simple NN (obtained from the World Wide Web)

A possible definition could be that a NN is a computational model consisting of interconnected artificial neurons organized into layers, capable of learning from data to make predictions or perform tasks based on patterns and examples. A N.N consists of nodes and connection between them. It has some unknown parameters called *weights* and *biases* that are multiplied and added to the input values correspondingly. Each *edge* has *weight*, each *neuron* has a *bias* term and each output is given after using the composition with a function of a type that is mentioned later. We use them to solve a learning problem, that consists of finding the optimal combination of *weights* so that the network function ϕ approximates a given function f as closely as possible. The function f is given, usually not in closed form but in data

points that come from some measurements[12]. In other words, the goal is to gain the optimized values that can fit our data the best way, using a method called *Backpropagation* which is nothing more than a practical application of the chain rule for derivatives[8]. The idea is to insert your data in the input nodes and finally get the desired result in the output nodes, while trying to minimize the error of our prediction, which can be defined by various functions called *Loss* or *Objective functions*.We set a upper bound for the error (a stopping criterion) that we demand to satisfy, else we stop the algorithm after a number of iterations called *epochs*. A typical *loss function* is the average the sum of the squared residuals (well known from least squares technique method). The layers of nodes between the input and output nodes are called *hidden layers*. We should add that in between the nodes we use a variety of *activation functions* that can be reshaped by the parameter values and then by using a linear combination of those, in order to fit the data into, for example, a curve. They enable the model to process complex data, by introducing non-linearity. A common example is the *ReLu* function which is given by the formula $f(x) = \max(0, x)$ and is well used in practice as it typically learns much faster in networks with many layers. It is also important to add that, due to their architecture that is based on non-linearity, NNs work better than linear classifiers for problems such as image recognition, as NNs are invariant to position or orientation shift. Linear classifiers can only divide their input data into two regions and this can accurately happen when the data are linearly separable, that is they can separated by a hyperplane. For example if two pictures were conceptually close, in the sense that one is first is a wolf and the latter a dog, a linear classifier probably could not distinguish between these two.

4.1.1 Neural Network training

Firstly, we shall want to compute parameters that their values are not too ”close”[1] so that we eliminate the chance that the N.N would have a bias giving all the parameters the same value. Furthermore, it is recommended that the starting (non-optimal) values of the weights are not close to zero. The reason is that, for example if we are using an activation function like the *sigmoid* $\left(\frac{1}{1+e^{-x}}\right)$, it becomes roughly linear, and hence the neural network converges into an approximately linear model. In addition, the network usually has many weights and that can lead to overfitting. A way to solve that issue is to add a regularization to the *loss function* with a parameter λ, whom most ideal value can be estimated using *cross validation*[2]. In order to have a better running NN, what users usually do, is to standardize all inputs to have mean zero and standard deviation one. All inputs are equally treated in the regularization process and we are allowed to choose uniformly the initial weights before training from a, usually, symmetric interval such as $[-0.7, +0.7]$[5]. An idea one might think of, when constructing a deep learning algorithm using a plethora of data, is to split our full data set to three different sets: a *train set* a *validation set* and a *test set*.

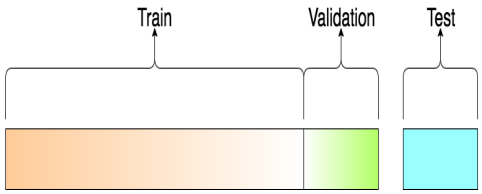


Figure 4.2: Data Set Partition (obtained from [1])

Then we try and train the algorithm with different choices of hyperparameters on the *training set*, evaluate on the *validation set* and now pick the set of hyperparameters which performs best in the *validation set*. Next, when we are done with the validation and there is nothing else to do, we take the model with the best performance on the *validation set* and run in **once** on completely unseen data, the *test set*. The score from the *test set* is the accuracy of the algorithm. It is really important to keep a strict separation

[1] One can define ”close” in various ways more compactly
[2] A well known technique to calculate hyperparameters

between then *validation* and *test* set. If we only split to *train* and *test set* and try training the N.N on the *train set*, then apply the trained NN on the *test set* and choose the method with the best hyperparameters on the *test data* will lead to inaccuracy problems, namely the test set will not be representative of how our method will do to unseen data. It is possible that we would have picked the right hyperparameters **only** for the according *test set*.

4.2 Convolutional Neural Networks

Since in this study we are interested in gaining information from spectrograms, we should also introduce the concept of *Convolutional Neural Networks* (CNN), which will be the kind of N.N that we are going to use. CNNs are designed to process data that come in the form of multiple arrays, such as a picture.

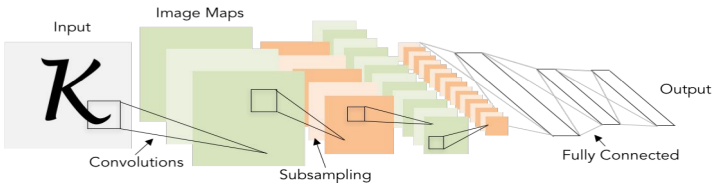


Figure 4.3: A CNN (obtained form [1])

CNNs take as input a picture that is represented as three dimensional matrix, namely it has width, height and depth that depends on the colors that constitute the picture. Then, we use filters that are three dimensional matrices with smaller dimensions than the image and we say that we *convolve* the image with the corresponding filter. That is, we slide the filter over the image with a step size called *stride* and compute the corresponding matrix multiplications[3]. When stride does not allow the filter to fit all the image, we usually add additional rows and columns with nulls, which means we increase the dimensions of the image. This procedure is called *zero padding*. Mathematically, the filtering operation performed is a discrete convolution, hence the name. The result is called *activation map*, which is a "three dimensional" matrix of depth one with different size than the original image. It is important to add that, the intensity of the elements of each filter used, are determined by *Backpropagation*, as well as the bias that is added right after the filtering, to each activation map, elementwise. In other words, the *weights* in a CNN are the elements of the filters. Sliding many filters on the image, we get a *convolutional layer* that consists of as many activation maps as the number of the filters we used and has the same width and height as every activation map. We can say that, a CNN is a sequence of Convolution Layers, interspersed with activation functions.[1]

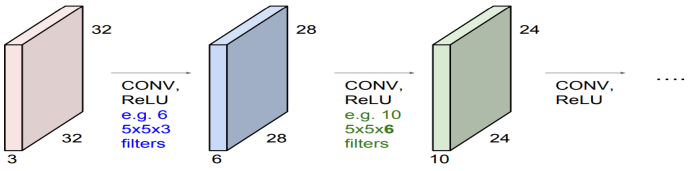


Figure 4.4: Structure of CNN, after each layer the dimension is reduced. (obtained form [1])

Usually we construct CNNs with such a way that the deeper it gets, it recognizes more details about the input image. This means that in the first layers the network checks really basic patterns and shapes, for example the first layer recognizes edges and corners, the second detects motifs based on the edges but independently of the edge position etc. As we move forward in the layers, the CNN recognizes more complex behavior. The reason is that, we increase the filter size in subsequent layers. It is important to add that *activation functions*, also, allow the CNN to capture patterns, edges, textures, and higher-level

[3] In strict mathematical terms, we take the inner product of the multidimensional arrays.

features in an image. During the filtering operation, after a couple of layers we apply the *pooling* which makes the representations smaller, more manageable, operates over each activation map independently and it tries to keep only the important information from the image. The pooling layer role is to merge similar features into one. A widely used technique is called *max pooling*, that keeps the maximum value of the image matrix, depending of course on the filter dimensions. This filter, usually does not overlap itself and its intuition is that *max pooling* selects the spots where the filter better matched the input image.[1]

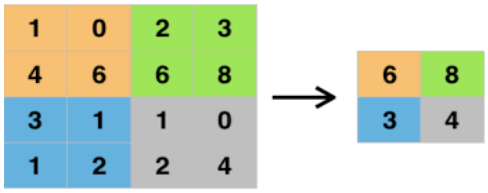


Figure 4.5: Max Pooling example (obtained form [1])

Finally we transform our last output to a *flattened array*, which then goes through a typical NN whose output represents ranking of possible outcomes or just values of some quantities the user is interested to obtain. This NN could contain *fully connected layers* (*dense layers*). A *fully connected layer* connects every neuron in the input to every neuron in output, while performing linear transformation to the input data.

There are some formulas that allow us to compute the dimensions of the output layer, after filtering, in several occasions. Namely, suppose that the image has dimensions $W_1 \times H_1 \times D_1$. Now, we set the following hyperparameters:

- Number of filters : K
- Spatial extent of the filter : F
- Stride : S
- Amount of zero padding : P

After filter application the dimensions are[4]:

- $W_2 = \frac{W_1 - F + 2P}{S} + 1$
- $H_2 = \frac{H_1 - F + 2P}{S} + 1$
- $D_2 = K$

Furthermore, the network gives $F \times F \times D_1$ weights per filter and total of $F \times F \times D_1 \times K$ weights and K biases.[1]

[4]The width and height are equally computed by symmetry

CHAPTER 5

Application

In this chapter we will apply the procedure described in the previous chapters to estimate the coefficients of the dispersion curves of a simulated recording of an acoustic signal due to Gaussian source.

5.1 The Inverse Problem

We will consider a marine environment with the first layer being water, the second fluid sediment and the third a semi-infinite fluid substrate. Our goal as presented in previous chapters is to identify the dispersion curves of a signal recorded in this environment by means of the coefficients of the splines that approximate their structure. Of course the signal, which in real cases and for the applications we have in mind, will be used as the tool for the estimation of the unknown environmental parameters, will be calculated with specific environmental parameters. Therefore, the actual dispersion curves of this signal will be known and our goal will be the estimation of the spline coefficients that describe these curves. In realistic applications the actual dispersion curves will not be known and the only tool in our hands for the estimation of spline coefficients would be the spectrogram of the measured signal.

5.2 Using CNN for Solving the Inverse Problem

The idea of using CNN for this inverse problem is that, in many applications exploiting the dispersion characteristics of a spectrogram, including those presented in the previous section, we can not compute the theoretical dispersion curves using the formula from Chapter 2. This happens, because we do not know the all the characteristics of the environment needed to calculate the dispersion curves. We only know the depth of the source and the receiver. CNNs can process complex pictures such as those corresponding to the energy dispersion provided by spectrograms and via training, to produce as output results important characteristics for the dispersion curves which are missing.

5.2.1 Data set creation

As we mentioned in the previous chapter, the mathematical tools that we are going to use are : spectrograms and cubic splines. Specifically, since we want the output of the CNN would be the spline coefficients describing the dispersion curves, the input of the CNN will be a data set that include **1120**

normalized spectrograms[1], for computation efficiency, using the formula:

$$\frac{X - \min(X)}{\max(X) - \min(X)} \tag{5.1}$$

that causes X to take values in the interval [0,1]. Symbolizing the CNN as a "function" F, then the spectrograms are the "values" X and we get $F(X) = Y$, where Y are the spline coefficients. The way we calculated the real coefficients is the following : For each signal and each mode[2], we set 8 specific nodes in the frequency domain, and retrieve via numpy the cubic polynomial coefficients in each interval that is defined by the nodes. The number 8 was chosen, because a single signal produces 224 coefficients, that is 224 CNN outputs for one signal, which is already a big number, let alone having a batch of signals. The structure of the Y is a set of matrices with dimensions 8×28, with each row referring to a specific mode of the signal and each column referring to the number of spline coefficients in a mode. It is important to add, that over the 1120 signals, we kept the *expected value* and *standard deviation* of each spline coefficient so that we denormalize the output of the CNN as with the usage of activation functions the outputs tend to be normalized.

5.2.2 Data set partitioning

We split our data set according to paragraph 4.1.1. Namely, we have a *train set* containing 60% of the total, a *validation set* containing 20% and a *test set* again containing 20%. Each set has its X values (spectrograms) and Y values (spline coefficient) that are going to be used when we calculate the values of the loss function. We note that the spline coefficients used are with boundary conditions that demand the first and second segment at a curve end to be the same polynomial. It is a good default when there is no information on boundary conditions.

5.2.3 Hyperparameters and algorithms

The hyperparameters used are shown in the following figure:

CNN Hyperparameters	
Epochs	**1000**
Learning rate	$\mathbf{10^{-3}}$
Batch size for training	**200**
Loss function	**Mean squared error**

Table 5.1: Hyperparameters

The algorithm chosen to optimize the weights and biases of the CNN is *Adam*. It is a combination of Stochastic Gradient Descent with acceleration (see Appendix B), which is many cases help to overcome local extrema or saddle points faster than vanilla Gradient Descent. There was no regularization used in the algorithm and the optimizer considers all parameter groups together and applies the same optimization strategy to all of them. In this case, the optimizer will compute a single set of adaptive learning rates and perform weight updates uniformly across all parameter groups.

5.2.4 CNN Architecture and Formulation

The network *architecture* consists of the following layers:

The first convolutional layer has 16 output channels, 3×3 kernel size, a stride of 1, and padding of 1. A max-pooling layer with a 2×2 kernel size and a stride of 2. The second convolutional layer has 32

[1] We produced 1120 signals, by defining search spaces on specific parameters of a known environment.
[2] We decided to keep the first 8 modes

output channels, 3×3 kernel size, a stride of 1, and padding of 1. Three fully connected (linear) layers with progressively decreasing output sizes. The weights initialization used is the default at pytorch: He initialization, which is typically implemented in N.Ns where the ReLu *activation function* is used.

The *formulation* of the CNN is:

The input is passed through the first convolutional layer followed by a ReLU activation function and then through max-pooling. Then, passed through the second convolutional layer with ReLU activation and max-pooling again.The corresponding output is flattened into a vector. Finally, the previous output is passed through the fully connected layers, which

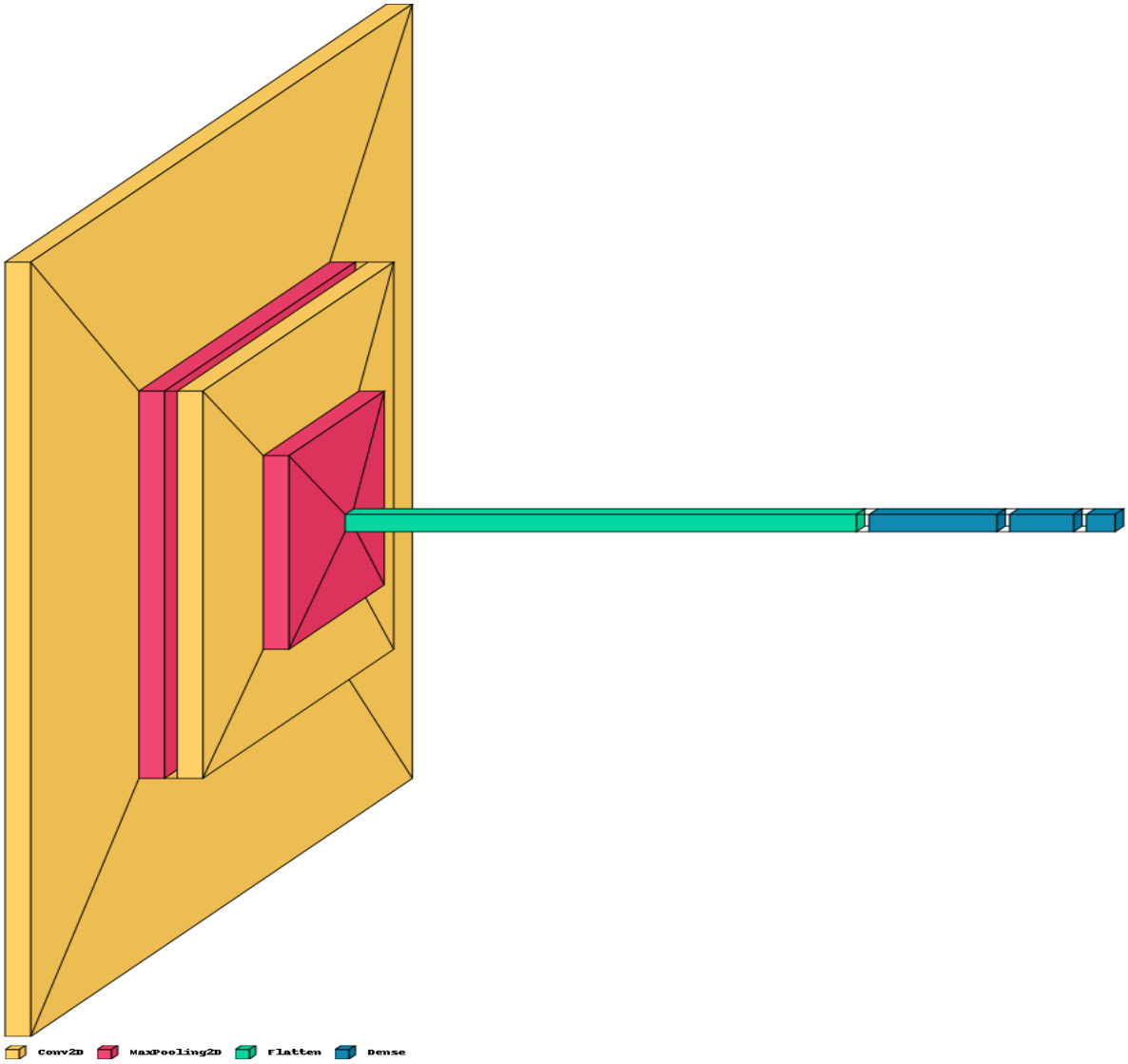


Figure 5.1: The CNN used

In addition, in the training phase we tried to optimize the weights and biases by trying minimize the quantity $||c_{pre} - c_{real}||_2$ or $||y_{pre} - y_{real}||_2$. The first means that we want to minimize the error between the spline coefficients and the latter that we try to minimize the values of the dispersion curves that results from using the corresponding coefficients. The second way is way more computationally expensive and in order to run in a reasonable amount of time, one have to use approximately 10 to 20 points in each interval where we find the y values. The results do differ, but only to the second or third decimal point. This makes the first attempt more reasonable and efficient. We also tried to make the CNN deeper using

4 or 5 layers, while using different weight initialization, in order to make more sensitive to details. The results were worse than the current CNN layering and we observed curve discontinuities not only in the last but also in the first 3-4 modes.

5.3 Spline Coefficients of the Dispersion Curves

Before presenting our results, let us remember that we rotated the *dispersion curves*so that we can apply the spline formulas. To retrieve the curve we want, we simply rotate again the results from the CNN. The change of the axes does not influence how *dispersion curves* characterize the environment. For example in a following *tomography* or *geoacoustic* inversion problem, where the desired result **m** are water sound velocity profile or sea floor characteristics, note that these are invariant to rotations of the corresponding **d** (in that case *dispersion curves*).

5.3.1 Test Case

We consider a problem of sea-bed classification being a typical inverse problem of acoustical oceanography. An acoustic source placed as some depth z_0 in the sea, emits acoustic signals that are recorded at some range R and depth z. For the application we have in mind the sound source will be considered with a gaussian source excitation function. The aim of the inversion procedure is to estimate the sea-bed properties expressed through the compressional wave velocities and the densities of the sediment layers. If the dispersion characteristics of the measured signal are to be used for the estimation of these parameters, the dispersion curves of the measured signal should be identified with good accuracy and this will be the goal of the inversion procedure described below. Note that the estimation of the sea-bed parameters is the subject of another inverse problem not considered in this work.

In the test case presented below, we simulate a *geoacoustic inversion* or *seabed classification* experiment, where an acoustic (broad-band) source with a Gaussian Source excitation function in the frequency domain with central frequency f_c= 150 Hz and bandwidth Δf= 100 Hz at depth z_0= 10 m emits a signal which is recorded by a hydrophone at depth z_r= 30 m and range R= 20000 m.

In order to apply a CNN for the estimation of the spline coefficients, we need a set of environments to be used at the training face. As already explained we will take into account some a-priori information on the construction of the sea-bed to define a search space of the 6 environmental parameters, among which the set of the environments will be considered. For this problem the set of environmental and operational parameters of the data set that will be used in CNN are presented in Table 5.2. Table 5.2 includes the limits of the sea-bed properties used to define the data set.

Parameters describing the marine environment
Water depth (**D**) **100 m**
Sediment thickness (**h**) **20 m**
Source Range (**R**) **20000 m**
Central frequency (f_c) **150 Hz**
Bandwidth frequency ($\triangle$**f**) **100 Hz**
Source/receiver depth ($\mathbf{z_0}$, $\mathbf{z_r}$) **(10,30) m**
Sound velocity profile in the water ($\mathbf{c_w}(z)$) $\mathbf{c_w}(0)$ **1500 m/sec** $\mathbf{c_w}(20)$ **1490 m/sec** $\mathbf{c_w}(100)$ **1510 m/sec**
Water density ($\rho_\mathbf{w}$) **1000 kg/m^3**
Sound velocity profile in the sediment ($\mathbf{c_b}(z)$) $\mathbf{c_b}(100)$ $\sim$ **[1600,1800] m/sec** $\mathbf{c_b}(120)$ **Same value as above**
Sediment density ($\rho_\mathbf{b}$) $\sim$ **[1300,1700] kg/m^3**
Sound velocity in the seabed ($\mathbf{c_{sb}}$) $\sim$ **[1600,2000] m/sec**
Seabed density ($\rho_\mathbf{sb}$) $\sim$ **[1600,1900] kg/m^3**

Table 5.2: Environments of the Data Set

Using the characteristics of the table, we created 1120 synthetic signals that consist our data set, which we partitioned as we mentioned before. We will show how the CNN process a random signal form the *test set*, which simulates a unknown signal. Specifically, we test the CNN on the first signal of the *test set* (the 696^{th} of the starting data set). The environment form which this signal comes from is the following:

Parameters describing the marine environment of Test case
Water depth (**D**) **100 m**
Sediment thickness (**h**) **20 m**
Source Range (**R**) **20000 m**
Central frequency (f_c) **150 Hz**
Bandwidth frequency ($\triangle$**f**) **100 Hz**
Source/receiver depth ($\mathbf{z_0}$, $\mathbf{z_r}$) **(10,30) m**
Sound velocity profile in the water ($\mathbf{c_w}(z)$) $\mathbf{c_w}(0)$ **1500 m/sec** $\mathbf{c_w}(20)$ **1490 m/sec** $\mathbf{c_w}(100)$ **1510 m/sec**
Water density (ρ_w) **1000 kg/m^3**
Sound velocity profile in the sediment ($\mathbf{c_b}(z)$) $\mathbf{c_b}(100)$ **1647 m/sec** $\mathbf{c_b}(120)$ **1647 m/sec**
Sediment density ($\rho_\mathbf{b}$) **1620 kg/m^3**
Sound velocity in the seabed ($\mathbf{c_{sb}}$) **1839 m/sec**
Seabed density ($\rho_\mathbf{sb}$) **1738 kg/m^3**

Table 5.3: Test Case Environment

We can visualize the environment in the next sketch:

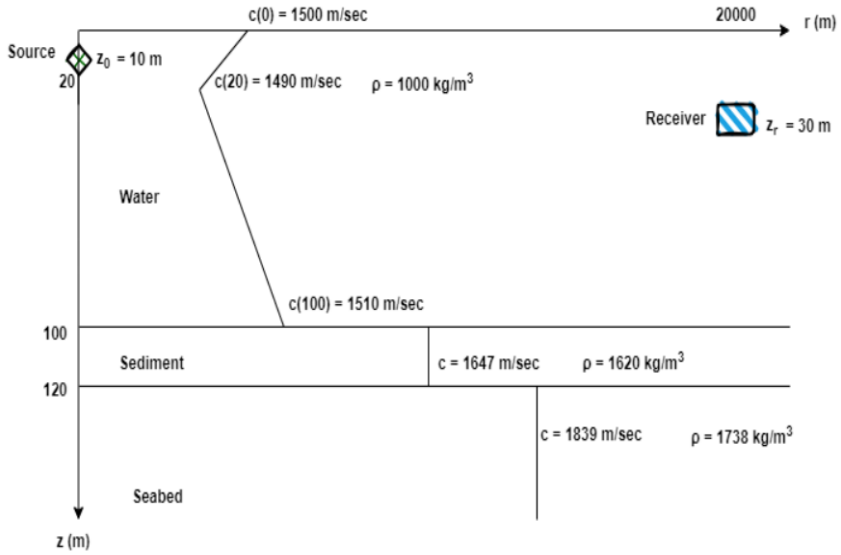
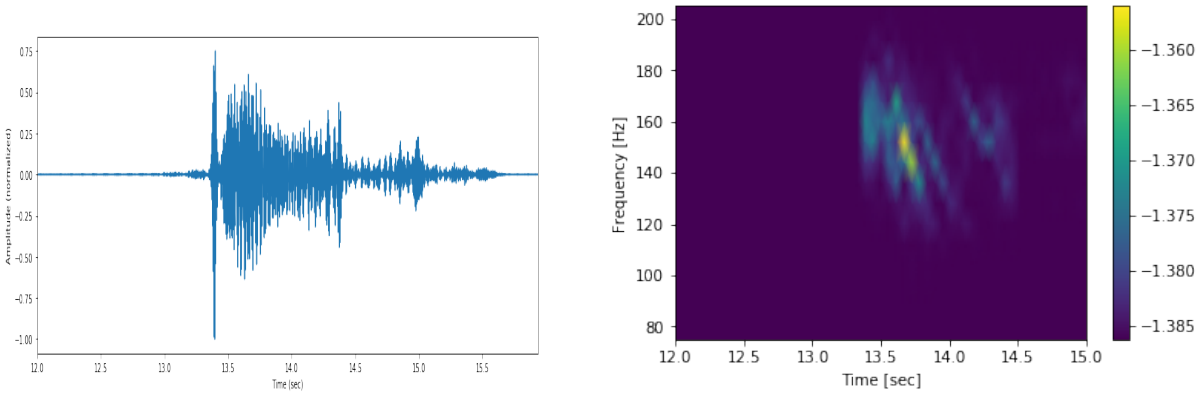


Figure 5.2: Test Case Environment Visualization

After we have passed the training phase, we use the spectrogram of the signal as the input in our CNN:

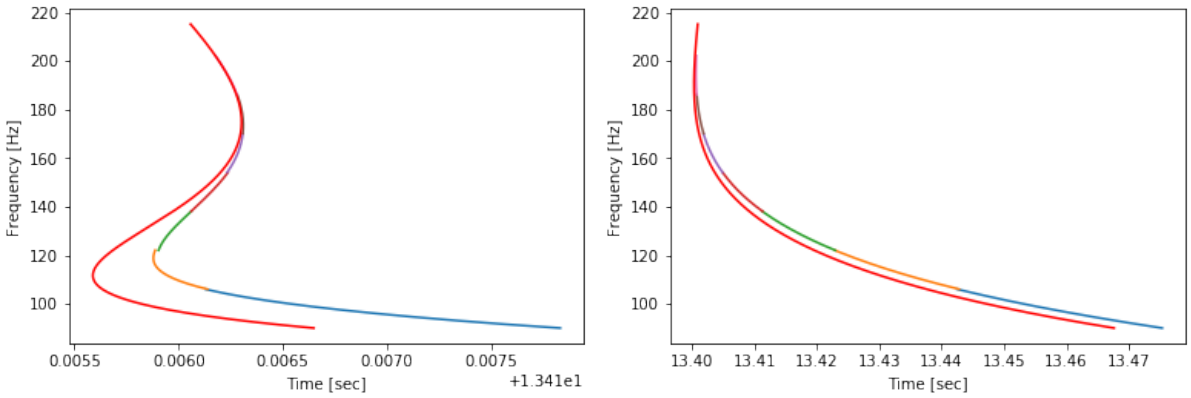


(a) Signal of the Test Case

(b) Spectrogram of the Test Case

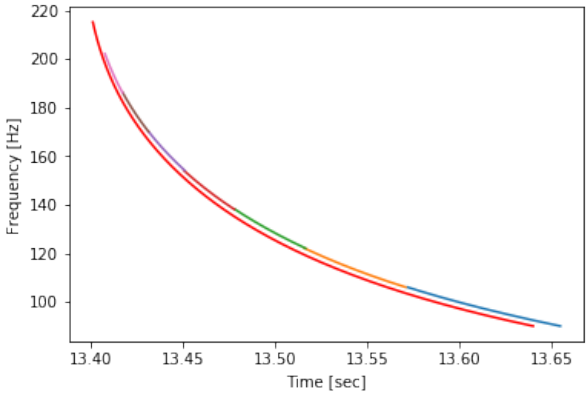
Figure 5.3: Test Case Signal and its Spectrogram

In Figure 5.4 we present the theoretical *dispersion curves* calculated with mode1 for the test signal and compare them with the *dispersion curves* given as output of the CNN:

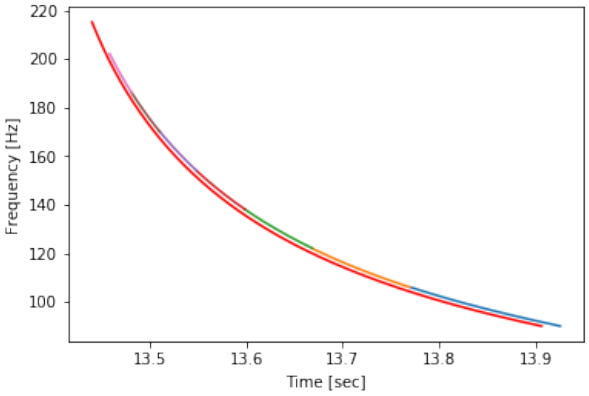


(a) Mode 1

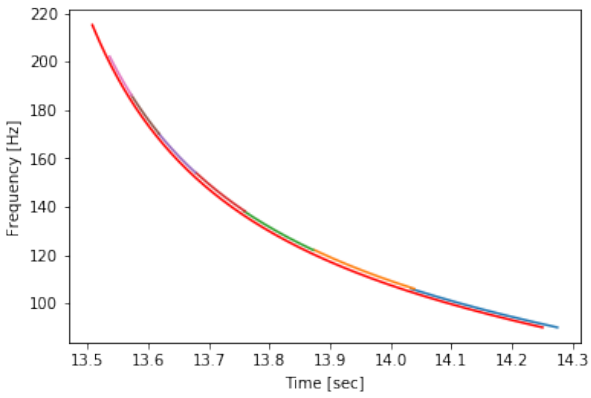
(b) Mode 2



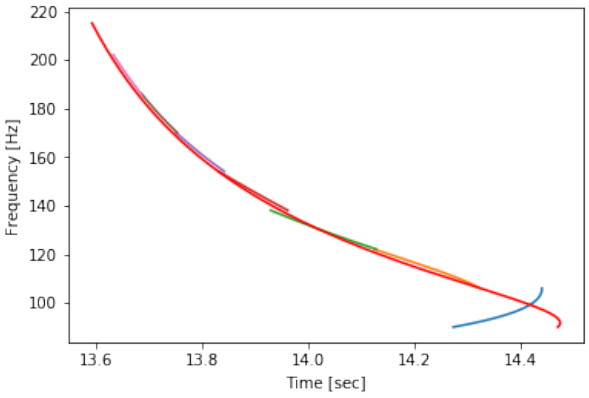
(c) Mode 3



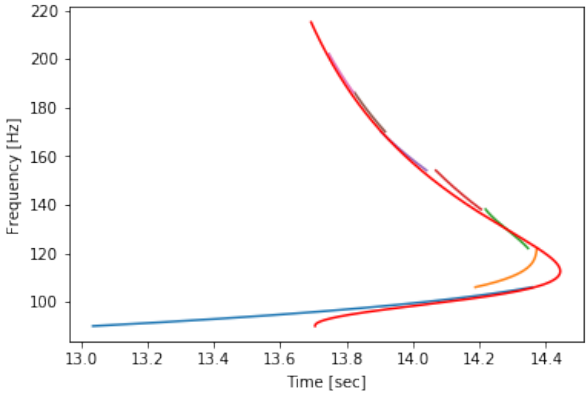
(d) Mode 4



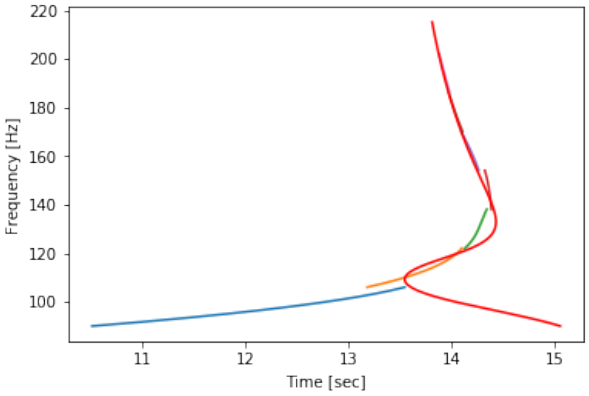
(e) Mode 5



(f) Mode 6



(g) Mode 7

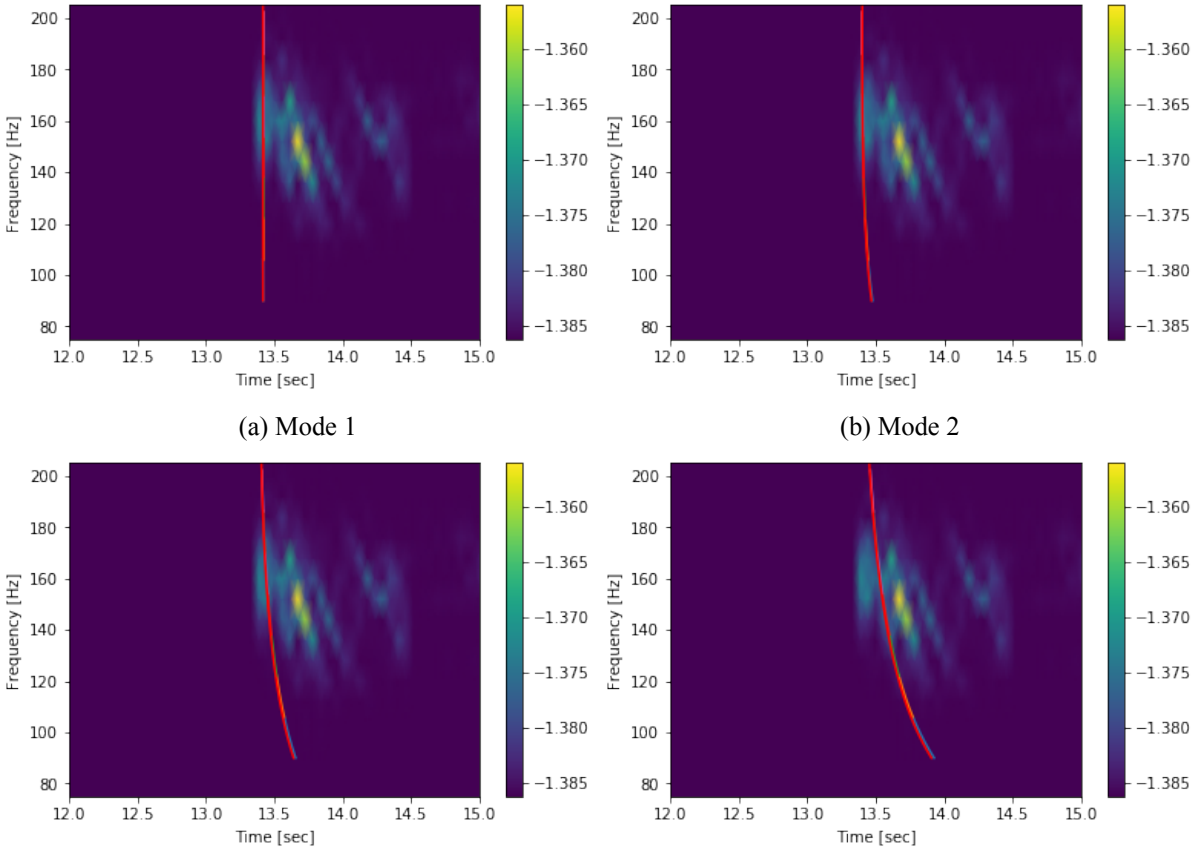


(h) Mode 8

Figure 5.4: Theoretical Dispersion Curves (Red) vs CNN Dispersion Curves (Colorful)

We have made visible with different colors the polynomials that are plotted in the **7** different intervals of the partition. We note that C.N.N *dispersion curves* have maximum frequency, around 200 Hz, whereas the theoretical around 220 Hz. This happens because of the nodes that we have taken to use the *cubic splines*. Namely, we used **8** nodes by running over all the available frequencies with step 128[3].

We can also see, that the C.N.N does not detect correctly the *airy phase* area, namely we have discontinuities as well as incorrect tracing for the early arrival times. Nevertheless, that does not seem to be a problem as we do not normally use these extreme areas from *dispersion curves* to retrieve information about the environment. As a result, we could just cut off the edges and proceed with the rest of the curves. We also notice, a similar issue at the first mode, where we might not have discontinuities as in the last modes, however the CNN curve does not approximate well at high curvature. This difference, does not influence the final results on the spectrogram, as the arrival times for the first mode are too small. So, as we will see, they seem compatible in the bigger picture. Now let us plot both kind of *dispersion curves* on the spectrogram to compare:



(a) Mode 1

(b) Mode 2

(c) Mode 3

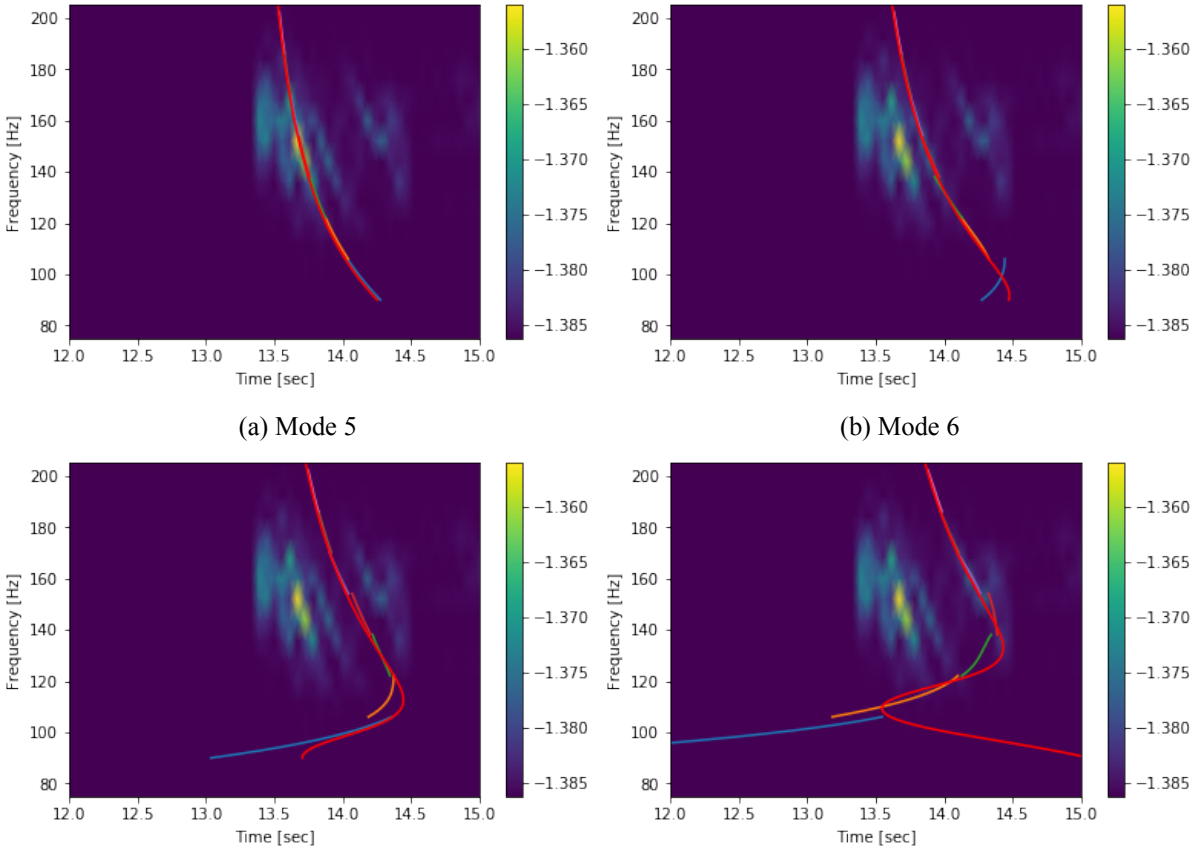
(d) Mode 4

Figure 5.5: Theoretical Dispersion Curves (Red) vs CNN Dispersion Curves (Colorful) on Spectrograms

CNN curves apply well on the spectrograms. We have similar results for the rest of the signals,as the different environments the data set is consisted of, have similar characteristics. Note that from the sixth mode and after, we can cut the values of the *dispersion curves* for the last interval of the partition to avoid the problematic points of the *Airy phase*. As we can see in the last two *dispersion curves*, we have satisfying results for frequencies higher than 120 Hz or 140 Hz. That is, by keeping the curve for frequencies higher than the previous mentioned, we have decent data to use them in the *geoacoustic*

[3]The nodes are: [90, 106.016, 122.032, 138.048, 154.064, 170.08 , 186.096, 202.112]

inversion problem.



(a) Mode 5

(b) Mode 6

(c) Mode 7

(d) Mode 8

Figure 5.6: Theoretical Dispersion Curves (Red) vs CNN Dispersion Curves (Colorful) on Spectrograms

.

CHAPTER 6

Conclusions

Summing up everything that has been stated so far, we initially described the modeling of wave propagation in ocean acoustics, namely we stated the *forward problem*. We then described the signal processing techniques that we used in our application and we did a brief introduction to discreet inverse problems and neural networks. We finally cited the results given by the convolutional neural network, that we constructed, and compared them with the theoretical *dispersion curves*. We observed that, the CNN curves fit in the right positions in the spectrogram, however we have some discrepancies mainly in the *Airy-phase* area. The results of the inverse problem are satisfying in the sense that we can cut pieces from the last dispersion curves for late arrival times, so that we keep only the segments that give the most accurate information. These dispersion curves can be very useful as *data* for other inverse problems such as *seabed classification* or *tomography*. In general in real world applications, we can not draw them theoretically due to lack of information. So, the CNN dispersion curves could be valuable for inversion problems in ocean acoustics. The CNN used might have a simple structure, however a deeper one seems to complicate the calculations and produce more inaccurate dispersion curves.

As we mentioned, the data set was created with 1120 signals (1120 different environments) using certain search spaces for the density and sound velocity in the sediment and substrate. The signals were basically variations of an original environment. So, in a sense the environments are very similar with each other. This leads to another approach, where we could make again a data set using 1120 signals, but this time we could create it using variations of two or three different environments. This could help the CNN to be more robust and give better results when we test it with unseen data. We could also implement the *loss function* of the CNN differently, in order to avoid discontinuities that occur mainly in the *Airy-phase*. Specifically the *loss function* used was the squared difference of *spline coefficients*: $||c_{pre} - c_{real}||_2^2$. We also tried the squared difference of the actual values of the dispersion curves: $||y_{pre} - y_{real}||_2^2$, but it was computationally expensive and did not have better results regarding the discontinuities. An idea is that, since the *cubic splines* calculate polynomials we can easily compute their derivatives, and since the function that can be represented via splines, must obey certain criteria like differentiability, we could additionally use the squared difference of the first and second derivative in a corresponding knot of the partition, while multiplying them by a constant that will have to be determined via trial and error or cross-validation. This implementation could solve the discontinuity problem at the knots and have smoother dispersion curves as the output of the CNN Based on this idea, the *loss function* would be written as:

$$||c_{pre} - c_{real}||_2^2 + [\quad \alpha||y_{pre} - y_{real}||_2^2 + \beta||y'_{pre} - y'_{real}||_2^2 + \gamma||y''_{pre} - y''_{real}||_2^2 \quad]_{\text{at a particular knot}}$$

where α, β and γ are to be determined.

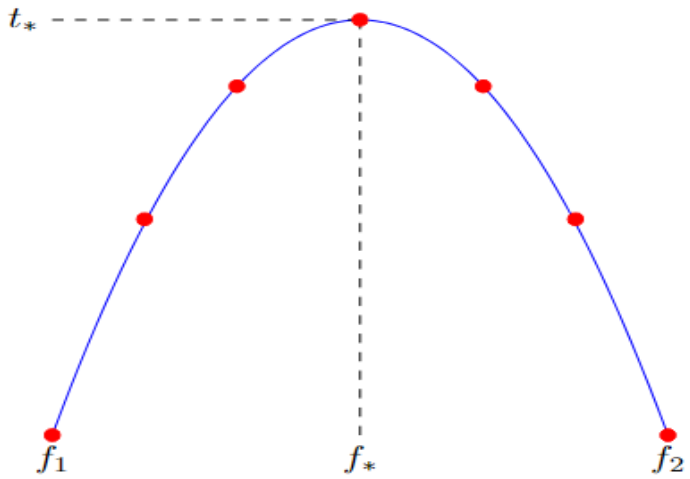


Figure 6.1: Sketch of a Dispersion Curve in frequency-time domain: We have two intervals $[f_1, f_*]$, $[f_*, f_2]$ that correspond to a cubic spline polynomial. The point (f_*, t_*) is the knot of the spline partition and the derivatives are calculated at this point.

A next step based on the idea of the thesis, is to use the results we got as the data for a seabed classification or tomography problem. However, the inversion would be interesting to be done via the CNN. That is, a next challenge is to create a CNN that computes the dispersion curves, when we use a spectrogram as input, and then does the inversion. Namely, its output is the marine environment characteristics that we seek, such as density and sound velocity.

Bibliography

[1] Deep learning for computer vision, stanford lecture notes - spring 2017.

[2] Charles Allan Boyles. Acoustic waveguides: applications to oceanic science. 1984.

[3] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[4] Peter E Hart, David G Stork, and Richard O Duda. *Pattern classification*. Wiley Hoboken, 2000.

[5] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[6] Finn B Jensen, William A Kuperman, Michael B Porter, Henrik Schmidt, and Alexandra Tolstoy. *Computational ocean acoustics*, volume 2011. Springer, 2011.

[7] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[8] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[9] William Menke. *Geophysical data analysis: Discrete inverse theory*. Academic press, 2018.

[10] Kraniauskas P. *From Fourier Integrals to Descreet Fourier Transforms.* 1992.

[11] S. Qian and D. Chen. Discrete gabor transform. *IEEE Transactions on Signal Processing*, 41(7):2429–2438, 1993.

[12] Raúl Rojas. *Neural Networks: A Systematic Introduction*. Springer-Verlag, Berlin, Heidelberg, 1996.

[13] Olga Sambataro, Costas Smaragdakis, and Michael Taroudakis. *A comparison of processing techniques applied to time-frequency represenation of acoustic signals intended for geoacoustic inversions*.

[14] Gilbert Strang. *Introduction to linear algebra*. SIAM, 2022.

[15] Michael I. Taroudakis. Program mode1, user's manual ite/iym. 1995.

[16] Γ.Δ. Ακρίβης and Β.Α. Δουγαλής. *Εισαγωγή στην Αριθμητική Ανάλυση*. Πανεπιστημιακές Εκδόσεις Κρήτης, 4 edition, 2015.

[17] Μιχάλης Ταρουδάκης. *Μαθηματική Μοντελοποίηση Κυματικής Διάδοσης στη Θάλασσα*, Πανεπιστημιακές Σημειώσεις, Πανεπιστήμιο Κρήτης 2020.

[18] Μιχάλης Ταρουδάκης. *Εισαγωγή στην Ακουστική Ωκεανογραφία*, Πανεπιστημιακές Σημειώσεις, Πανεπιστήμιο Κρήτης 2020.

APPENDIX A

Splines Theory

Definition A.0.1. (Smooth Spline) Let [a,b] $\subset \mathbb{R}$, $\Delta : a = x_0 < x_1 < ... < x_n = b$ partition of [a,b] and m$\in\mathbb{N}$. The elements of (linear) space

$$S_m(\Delta) \coloneqq \{s \in \mathcal{C}^{m-1}[a,b] : \quad s|_{[x_i,x_{i+1}]} \in \mathbb{P}_m, \quad i = 0, ..., n-1\}$$

are called (*polynomial*) *splines* of degree m (in respect of Δ), where $\mathbb{P}_m$ is the space of polynomials of degree m[16].

A.1 Interpolation with piece-wise linear functions

Let [a,b] $\subset \mathbb{R}$, $\Delta : a = x_0 < x_1 < ... < x_n = b$ partition of [a,b] and $S_1(\Delta)$ the corresponding linear space of splines degree one, namely the space of continuous at [a,b] and piece-wise continuous (in respect of Δ) linear functions. We define the functions $\phi_0, ..., \phi_n$ with the formulas:

$$\phi_0(x) \coloneqq \begin{cases} -\frac{x-x_1}{x_1-x_0}, & \text{if } x_0 \le x \le x_1 \\ 0 \quad , & \text{otherwise} \end{cases} \tag{A.1}$$

$$\phi_i(x) \coloneqq \begin{cases} \frac{x-x_{i-1}}{x_i-x_{i-1}} \quad , & \text{if } x_{i-1} \le x \le x_i \\ -\frac{x-x_{i+1}}{x_{i+1}-x_i}, & \text{if } x_i \le x \le x_{i+1}, \qquad i = 1, ..., n-1 \\ 0 \quad , & \text{otherwise} \end{cases} \tag{A.2}$$

and

$$\phi_n(x) \coloneqq \begin{cases} \frac{x-x_{n-1}}{x_n-x_{n-1}}, & \text{if } x_{n-1} \le x \le x_n \\ 0 \quad , & \text{otherwise} \end{cases} \tag{A.3}$$

It can be verified that, the set $\{\phi_0, ..., \phi_n\}$ is a *base* of $S_1(\Delta)$, so this space has dimension $n+1$. It is important that, the *support* (the smallest closed interval out of which $\phi_i, i = 0, ..., n$ are zero) of the *base functions* should be small for applications.

For every function $f \in \mathcal{C}[a,b]$ there is exactly one function $s \in S_1(\Delta)$, such that:

$$s(x_i) = f(x_i), \quad i = 1, ..., n \tag{A.4}$$

The function s is given by the formula:

$$s = \sum_{i=0}^{n} f(x_i)\phi_i \tag{A.5}$$

If $p_i \in \mathbb{P}_1$ is the linear polynomial of f interpolation at points x_i and x_{i+1}, that is if:

$$p_i(x) = f(x_i) + \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}(x - x_i) \tag{A.6}$$

then it holds that $s(x) = p_i(x)$ for $x \in [a, b]$. In this case, the interpolation is an entirely local situation, that is the values of s in $[x_i, x_{i+1}]$ depend only on f values at the points x_i and x_{i+1}.

Theorem 3. *(Error estimation of interpolation with piece-wise linear functions) Let $f \in \mathcal{C}^2[a, b]$, $\Delta : a = x_0 < x_1 < ... < x_n = b$ partition of [a,b] and $s \in S_1(\Delta)$ the interpolation function of f at the points $x_1, ..., x_n$. If $h_i \coloneqq x_i - x_{i-1}$ and:*

$$h \coloneqq \max_{1 \le i \le n} h_i \tag{A.7}$$

then it holds:

$$||f - s||_\infty \le \frac{h^2}{8}||f''||_\infty \tag{A.8}$$

A.2 Cubic splines interpolation

Definition A.2.1. (Cubic Splines) Let $n \in \mathbb{N}$, $h \coloneqq \frac{b-a}{n}$ and $\Delta_h : a = x_0 < x_1 < ... < x_n = b$ a uniform partition[1] of [a,b] with step h, that is $x_i = a + ih$, $i = 0, ..., n$. We consider the space $S_3(\Delta_h)$, namely the linear space of functions that are twice differentiable at $[a, b]$ and that are cubic polynomials in every subinterval of $[x_i, x_{i+1}]$ of the partition Δ_h. These functions are called *cubic splines*.

The function s is a different cubic polynomial in every subinterval $[x_i, x_{i+1}]$, $0 \le i \le n - 1$, so the construction of s requires $4n$ coefficients to be found in every subinterval. From:

$$s(x_i) = f(x_i), \quad i = 0, ..., n \tag{A.9}$$

we have n+1 equations.The internal nodes of Δ_h are: x_i ,$i = 1, ...n - 1$. We know that $s \in S_3(\Delta_h)$, $s \in \mathcal{C}^2[a, b]$, so from the continuity of s, s' and s'' at x_i, $i = 1, ..., n - 1$ we have $3n - 3$ equations. We have $4n - 2$ in total and we need two more so that we can uniquely define a $s \in S_3(\Delta_h)$. These two equations will be determined from the boundary conditions in a and b.

Theorem 4. *(Cubic Splines Interpolation) Let $f \in \mathcal{C}^1[a, b]$, $n \in \mathbb{N}$, $h \coloneqq \frac{b-a}{n}$ and $\Delta_h : a = x_0 < x_1 < ... < x_n = b$ a uniform partition of $[a, b]$ with step h. Then there is exactly one function $s \in S_3(\Delta_h)$, such that:*

$$\begin{cases} s(x_i) = f(x_i), & i = 0, ..., n \\ s'(x_0) = f'(x_0) \\ s'(x_n) = f'(x_n) \end{cases} \tag{A.10}$$

Theorem 5. *(Cubic Splines Interpolation)[2] Let $f \in \mathcal{C}^2[a, b]$, $n \in \mathbb{N}$, $h \coloneqq \frac{b-a}{n}$ and $\Delta_h : a = x_0 < x_1 < ... < x_n = b$ a uniform partition of $[a, b]$ with step h. Then there is exactly one function $s \in S_3(\Delta_h)$, such that:*

$$\begin{cases} s(x_i) = f(x_i), & i = 0, ..., n \\ s''(x_0) = f''(x_0) \\ s''(x_n) = f''(x_n) \end{cases} \tag{A.11}$$

[1] For simplification usually we use uniform partition. Generalization for non-uniform partition can be similarly defined

[2] In the bibliography of numerical analysis one can see more ways to define the values of s' or s'' in the boundary

Theorem 6. *(Error estimation of cubic splines interpolation) Let* $\Delta : a = x_0 < x_1 < ... < x_n = b$ *a partition of* $[a, b]$*,*

$$h \coloneqq \max_{1\leq i\leq n}(x_i - x_{i-1}), \quad M \coloneqq \frac{h}{\min\limits_{1\leq i\leq n}(x_i - x_{i-1})} \tag{A.12}$$

$f \in \mathcal{C}^4[a, b]$ *and* $s \in S_3(\Delta)$ *a spline that satisfies* (A.10)*. Then, there are* C_m *coefficients,* $m = 0, 1, 2, 3$*, independent of* f *and* h*, such that :*

$$||f^{(m)} - s^{(m)}||_\infty \leq C_m h^{4-m}||f^{(4)}||_\infty \tag{A.13}$$

Completely similar estimations hold for spine interpolation with second derivative boundary conditions.

APPENDIX B

Additional information for Neural Networks

B.1 Gradient Descent and its variations

In *optimization theory* or *mathematical programming*, the main goal is to minimize (or maximize) a quantity, a function. There are a lot of different kind of optimization subfields such as *convex programming* where the objective function is convex, or *nonlinear programming* where the objective function or the parameters have nonlinear characteristics. A common method used in a lot of these subfields to find the minima (mixama) is *Gradient Descent (Ascent)*. We are going to focus on *Gradient Descent* as in our problem we search the minima of a *loss function*.

B.1.1 Gradient Descent

The concept of the following algorithm is to follow the flow of gradient to gradually move to the lowest point of the graph. As we are getting closer to the minima of the function the step we are making is getting smaller.

Algorithm 1 Gradient Descent

Require: f(θ): Convex and differentiable objective function with parameters θ
Require: η: Learning rate
Require: θ_0: Initial parameter vector
Require: M: Max Iter
Require: ϵ: Step size threshold
Require: $\eta \nabla_\theta f(\theta_0)$: Initial Step size
while Step size < ϵ **do**
    Step size=$\eta \nabla_\theta f(\theta_n)$
    θ_{n+1}=θ_n-Step size
    if Iterations$\geq$M **then**
        Break
    end if
end while
argmin $(f(\theta)) \longleftarrow \theta_{N+1}$
θ
return argmin $(f(\theta))$
θ

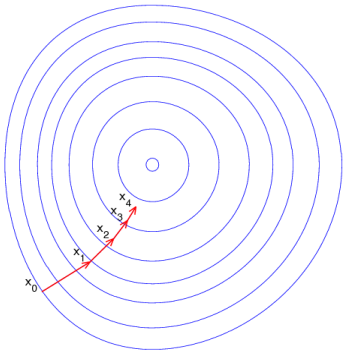


Figure B.1: Vanilla Gradient Descent (obtained from the World Wide Web)

B.1.2 Adam Algorithm

This method is based on stochasticity, that is it chooses a random batch of data in each iteration for computational efficiency and also works with stochastic *objective functions*. It also uses the idea of *accelerated gradient descent* and *Nesterov algorithm*[1] to skip any local extrema or saddle points. Many, consider this method the best for optimizing an *objective function*.

Algorithm 2 Adam Algorithm[7]

Require: α: Step size
Require: $\beta_1, \beta_2 \in [0, 1]$: Exponential decaying rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \longleftarrow 0$ (Initialize 1^{st} moment vector)
$v_0 \longleftarrow 0$ (Initialize 2^{st} moment vector)
$t \longleftarrow 0$ (Initialize time step)
while θ_t not converged **do**
  $t \longleftarrow t + 1$
  $g_t \longleftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at time step t)
  $m_t \longleftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
  $v_t \longleftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
  $\hat{m}_t \longleftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
  $\hat{v}_t \longleftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
  $\theta_t \longleftarrow \theta_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

B.2 Backpropagation

Backpropagation[12] is a main procedure implemented in NNs, that characterizes their structure and functionality, but it is based on simple calculus: the *chain rule* and also gradient descent or its variations. Its usage is to optimize all the weights and biases simultaneously in the N.N during the training phase. Conceptually, *Backpropagation* starts with the last parameter and estimates all the others backwardwise. Now, we are going to introduce a more compact symbolism for gradient descent in order to describe *Backpropagation* algorithm. Let the *weights* be $\boldsymbol{w}$ and $f(\boldsymbol{w})$ the *loss function*. The weights are initialized with random values and the variation is done in the direction where the error is reduced or equivalently

[1] The interested reader can search in the bibliography for more details

where the gradient tends to zero:

$$\Delta \boldsymbol{w} = -\eta \nabla_{\boldsymbol{w}} f(\boldsymbol{w}) \tag{B.1}$$

where η is the learning rate. We will consider, a NN c outputs, n_H intermediate neurons, three hidden layers and d inputs. The weight $w_{ij}^{(1)}$ symbolizes the weight between the input site i and the hidden unit j, whereas $w_{ij}^{(2)}$ the weight between hidden unit i and output unit j[12].

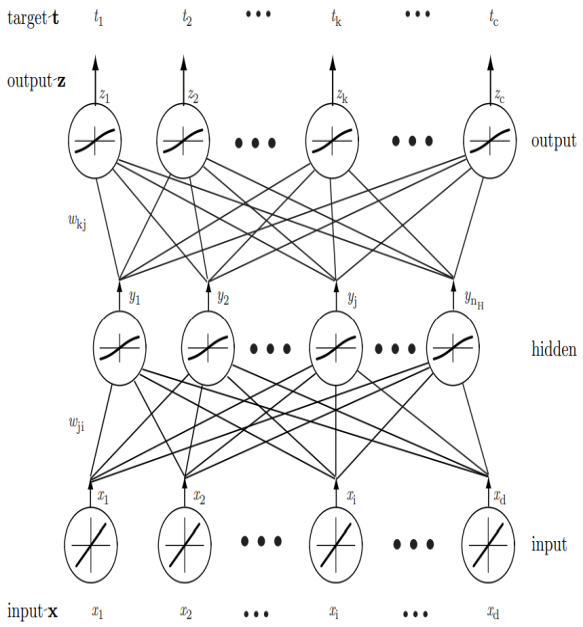


Figure B.2: Scheme of the Neural Network with input **x** output of hidden layers **y** and output **z** (obtained from [12])

Each of the hidden unit takes the inner product of the input with the corresponding weights, that is

$$net_j = \sum_{i=1}^{d} w_{ij}^{(1)} x_i \tag{B.2}$$

through the hidden units will give:

$$y_j = F(net_j) \tag{B.3}$$

where F is an activation function. In the same manner we work with the with the outputs of the hidden layers and the outputs of the NN. Componentwise, B.1 can be written as:

$$\Delta w_{ij} = -\eta \frac{\partial f(\mathbf{w})}{\partial w_{ij}} \tag{B.4}$$

Using the *chain rule* we compute:

$$\frac{\partial f(\mathbf{w})}{\partial w_{ij}} = \frac{\partial f(\mathbf{w})}{\partial net_i} \frac{\partial net_i}{\partial w_{ij}} = -\delta_i \frac{\partial net_i}{\partial w_{ij}} \tag{B.5}$$

where δ_i is called *sensitivity*[4], describes how the overall error changes with the unit's activation and it is defined as:

$$\delta_i \equiv -\frac{\partial f(\mathbf{w})}{\partial net_i} \tag{B.6}$$

Now we can show two *Backpropagation* batch algorithms:

Algorithm 3 Batch Backpropagation[4]

Require: initialize network topology (# hidden units), $\boldsymbol{w}$, criterion ϵ, η, $r \leftarrow 0$

repeat
  $r \leftarrow r + 1$
  $m \leftarrow 0$; $\Delta w_{ij}^{(1)} \leftarrow 0$; $\Delta w_{ij}^{(2)} \leftarrow 0$
  repeat
    $m \leftarrow m + 1$
    $\boldsymbol{x}_m \leftarrow$ select pattern
    $\Delta w_{ij}^{(1)} \leftarrow \Delta w_{ij}^{(1)} + \eta\delta_j x_i$; $\Delta w_{ij}^{(2)} \leftarrow \Delta w_{ij}^{(2)} + \eta\delta_j y_i$
  until $m = n$
  $w_{ij}^{(1)} \leftarrow w_{ij}^{(1)} + \Delta w_{ij}^{(1)}$; $w_{ij}^{(2)} \leftarrow w_{ij}^{(2)} + \Delta w_{ij}^{(2)}$
until $\nabla f(\boldsymbol{w}) < \epsilon$
return $\boldsymbol{w}$

Algorithm 4 Stochastic Backpropagation[4]

Require: initialize network topology (# hidden units), $\boldsymbol{w}$, criterion ϵ, η, $m \leftarrow 0$

repeat
  $m \leftarrow m + 1$
  $\boldsymbol{x}_m \leftarrow$ randomly chosen pattern
  $w_{ij}^{(1)} \leftarrow w_{ij}^{(1)} + \eta\delta_j x_i$; $w_{ij}^{(2)} \leftarrow w_{ij}^{(2)} + \eta\delta_j y_i$
until $\nabla f(\boldsymbol{w}) < \epsilon$
return w